ODE Guidance for the Content of Premarket Submission for Medical Devices Containing Software

Draft Document

This guidance document is being distributed for comment purposes only.

CDRH Software Task Force Group
Center for Devices and Radiological Health

Draft released for comment on: September 3, 1996

Comments and suggestions regarding this draft document should be submitted within 120 days of the above release date to Joanna H. Weitershausen, HFZ-450, Office of Device Evaluation, 9200 Corporate Blvd., Rockville, Maryland 20850. Comments and suggestions received after this date may not be acted upon by the Agency until the document is next revised or updated. For questions regarding this draft document, contact Joanna H. Weitershausen at (301) 443-8609.

U.S. DEPARTMENT OF HEALTH AND HUMAN SERVICES
Food and Drug Administration

Center for Devices and Radiological Health

CDRH Software Task Force

This document was prepared and reviewed by the FDA Center for Devices and Radiological Health (CDRH) Software Task Force, which included representatives from the following offices:

Office of Compliance and Surveillance (OCS)
E. Stewart Crumpler, OC Software Cross-cutter, OC/DOE3

Office of Device Evaluation (ODE)
Laura Byrd, Electrical Engineer, ODE/DRAERD
James M. Cheng, Computer Scientist, ODE/DCRND
Robert DeLuca, ODE/DGRD
*Joanna Haywood, Electrical Engineer, ODE/DCRND
Jodi Nashman, Biomedical Engineer, ODE/DGRD
Neil Ogden, Biomedical Engineer, DGRD/GSDB
Kimberly Peters, Biomedical Engineer, ODE/DDIGD
Ralph E. Shuping, DSc., Health Physicist, ODE/DRAERD
Thomas E. Simms, Biologist, ODE/DCLD
Morris Waxler, PhD., Psychologist, ODE/DOD
**David A. Zier, Electrical Engineer, ODE/DCRND

Office of Health Industries Programs (OHIP)
Peter B. Carstensen, Human Factors Engineer, OHIP/DDUPSA
Charles R. Sawyer, PhD., Human Factors Scientist, OHIP/DDUPSA

Office of Surveillance and Biometrics (OSB)
Sharon Dillard
Isaac Hantman, Biomedical Engineer, OSB/DSS

Office of Science & Technology (OST)
Joseph Jorgens, OST/DECS
**Debra S. Herrmann, Computer Scientist, OST/DECS

FDA Office of Regulatory Affairs (ORA/DFI)
David Bergeson, Consumer Safety Officer
Denise Dion, Consumer Safety Officer

FDA Center for Biologics Evaluation and Research (CBER)
John Capen, Operations Research Analyst, (CBER/OBRR)

FDA Center for Drug Evaluation and Research (CDER)
Charles Snipes, PhD., Pharmacologist (CDER/OC)

* Task Force chairperson
** Former Task Force co-chair

Independent Reviewers

The following individuals and organizations, representing academia, industry, and a federal regulatory agency, provided invaluable input during the informal pre-review of draft version 1.1 of this guidance. Participation in the pre-review of the guidance does not necessarily imply approval by such reviewers or by the organizations with which they are affiliated.

1. Michael P. DeWalt, National Resource Specialist -- Software
   Federal Aviation Administration (FAA)
   ANM-106N
   1601 Lind Avenue SW
   Renton, WA 98055
   206.227.2762

2. Dr. Martin D. Fraser
   Department of Mathematics and Computer Science
   Georgia State University
   Atlanta, GA 30303-3083
   404.651.2245

3. Dr. Vijay Vaishnavi
   Department of Computer Information Systems
   Georgia State University
   P.O. Box 4015
   Atlanta, GA 30302-4015
   404.651.3891

4. Dee Simons, Vice President of Science & Technology
   Health Industries Manufacturers Association (HIMA)
   1200 G Street NW, Suite 400
   Washington, DC 20005-3814
   202.783.8700

5. Gary LeBlanc, Director of Quality Assurance/Regulatory Affairs
   Indiana Medical Device Manufacturers Council (IMDMC)
   1908 East 64th Street, South Drive
   Indianapolis, IN 46220-2186
   317.257.8558

6. Robert G. Britain, Vice President, Medical Products
   National Electrical Manufacturers Association (NEMA)
   1300 North 17th Street, Suite 1847
   Rosslyn, VA 22209
   703.841.3241

7. Dr. Ugo Buy
Electrical Engineering and Computer Science Department
University of Illinois (M/C 154)
Chicago, IL 60607
312.413.2296

8. Dr. Carmen Trammell
Department of Computer Science
University of Tennessee
107 Ayres Hall
Knoxville, TN 37966
423.974.5784

Document Update and Revision Log

This document is issued on double sided 3-hole punch paper with tabs between sections. It is intended to be used in a loose-leaf notebook. This will facilitate the issuance of periodic additions, change pages, and updates to accommodate the rapid evolution of software technology. The log below identifies the most current configuration of the document. For easy identification, the version number and publication date are printed on the top right and top left corners of each page. Planned additions, change pages, and updates to the document will be announced at the annual AAMI/FDA International Standards Conference and through the new HIMA-NET. Copies of the document and the most current revision log are available from FDA/CDRH/OHIP Division of Small Manufacturers Assistance, 1.800.638.2041.

After the first official version of this document is issued the implementation date will be effective 90 days later. This phase-in period will allow industry an opportunity to incorporate the guidance into its submissions practices in order to ensure the best possible submissions which can be efficiently and effectively reviewed by FDA. Until the guidance goes into effect, both FDA reviewers and industry should continue to use the current version of the guidance.

Version	Dated	Section(s)	Pages	Comments
draft 1.0	11/16/95	all	all	initial CDRH review
draft 1.1	12/15/95	all	all	review by CDRH and independent reviewers
working drafts	3/04/96	all	all	Taskforce comments
	3/13/96	all	all	CDRH comments
draft 1.2	5/03/96	all	all	CDRH comments
draft 1.3	8/12/96	all	all	formal comment/review per FR notice

1.0	2/03/97	all	all	first official version issued
1.0	5/03/97	all	all	guidance takes effect

Table of Contents

APPENDICES

List of Figures

SECTION 1. Introduction

1.1 Purpose.

This document provides guidance on the regulatory review of premarket medical device software submissions. It replaces the "Reviewer Guidance for Computer Controlled Medical Devices Undergoing 510(k) Review" issued in 1991. This guidance discusses the key elements reviewers look for in a premarket medical device software submission; thereby providing a common baseline from which both manufacturers and scientific reviewers can operate. This guidance is also intended to further the understanding of software engineering practices, quality, reliability, and safety; mainly by referring to existing standards (Appendix D), textbooks (Appendix E), and generic software development tutorial (Appendix A).

1.2 Background.

The primary motivation for developing this new guidance was to clarify the existing guidance. FDA has four years experience using the existing guidance. Feedback from both manufacturers and scientific reviewers has been incorporated into the new guidance. By clarifying the guidance, the Agency hopes to receive a larger percentage of complete premarket submissions the first time around. This will avoid the need for additional information requests which are time and resource consuming for both FDA and manufacturers. In addition, the guidance has been updated to be consistent with emerging international consensus standards such as International Electrotechnical Commission (IEC) 601-1-4 and International Organization for Standardization (ISO) 9001 and ISO 9000-3, and the proposed new Good Manufacturing Practices (GMPs).

A preview of the new guidance was presented May 17, 1995, during the Indiana Medical Device Manufacturers Council (IMDMC) Seminar on the Development of Medical Device Software in a Regulated Environment and again on September 21, 1995 during the 19th Annual Regulatory Affairs Professional Society (RAPS) Exhibition and Conference. In addition, the independent reviewers listed on page iii provided invaluable feedback during the informal review of draft version 1.1 of this document. A short course, explaining how to use this guidance document is being coordinated through the FDA Center for Devices and Radiological Health (CDRH) Staff College.

1.3 Scope.

The new software guidance applies to all types of premarket submissions: premarket notifications (510(k)), premarket applications (PMAs), and investigational device exemptions (IDEs). The previous guidance indicated that is was for 510(k) submissions. However, in practice it was used for PMA and IDE submissions as well since no software guidance existed for these types of submissions. Differences in the types of information to be submitted and how it is evaluated are noted in the text.

In general, this guidance applies to all software which is used in medical devices. This includes:

1) software which is embedded in medical devices, i.e. firmware;

2) software which requires operator interaction, such as setting parameters or selecting modes; and

3) software accessories as defined in the draft software policy.

(Note: these categories are not necessarily mutually exclusive.) While the same development lifecycle and risk management activities apply, from a premarket regulatory perspective at present it excludes pure hospital information systems, such as billing records, and manufacturing process control software. (See B.13)

1.4 Intended Audience.

This guidance is intended for use by scientific reviewers within the FDA CDRH Office of Device Evaluation (ODE) who review medical device software, FDA Office of Regulatory Affairs (ORA) Investigators, and the medical device industry.

1.5 Document Organization.

This document is divided into four normative sections and six informative appendices. The terms normative and informative are used the same way as in international standards: normative - strongly encourage; informative - useful background information.

- Section 1, which is normative, describes the purpose, background, scope, intended audience, document organization, relationship to other documents, and terminology.

- Section 2, which is normative, discusses software risk management activities.

- Section 3, which is normative, explains the level of concern determination and how it relates to the documentation in and review of a premarket submission.

- Section 4, which is normative, identifies key informational elements of a premarket submission.

- As a supplement to Section 2, Appendix A provides more detail about development life cycle activities and software engineering practices. This appendix is informative.

- Special topics and unique technological issues currently facing the FDA and industry are examined in Appendix B. This appendix is informative.

- Appendix C provides a sample checklist and common requests for reviewers and manufacturers to use during the premarket review process. This appendix is informative.

- Appendices D and E cite current relevant national and international consensus standards and texts related to software engineering, quality assurance, risk management and lifecycle methodologies. Reviewers and manufacturers should refer to these sources for more in-depth information, since that information is not duplicated in this guidance. These appendices are informative.

- The "FDA Glossary of Computerized System and Software Development Terminology", developed by the ORA Division of Field Investigations (DFI), appears in Appendix F. Since many of the terms used in the software field have multiple definitions, this appendix is designed to provide a reference for terms only as used in this document. This appendix is informative.

1.6 Relationship to Other Documents.

This publication is a guidance document, not a standard. FDA and device designers use guidance documents to provide a means, among possible others, to meet regulations and policy, as shown in Figure 1-1. National and international consensus standards, such as those cited in Appendix D, are viewed as tools for demonstrating compliance with regulations.

Since this guidance addresses a cross-cutting issue, it is intended to complement device specific guidance by providing additional detailed software information.

1.	Mission	--->	Protect the public health
2.	Law(s)	--->	SMDA 1990, MDA 1992, ...
3.	Regulation(s)	--->	21 CFR xxx
4.	Policy/Guidance	--->	ODE Guidance for the ...
5.	Standards	--->	IEC 601-1-4, ...

Figure 1-1. U.S. Regulatory Hierarchy for Medical Device Software.

1.7 Terminology.

FDA guidance documents by definition do not establish legally binding requirements. A guidance document is published as providing a means, but not the only means, of showing compliance to regulations which are very general in nature. The terms "should" and "must" are used in this document. However, they should be interpreted as "should" or "must" from a theoretical or technical perspective -- not a legal perspective.

SECTION 2. Software Risk Management Activities

This section provides an overview of risk management activities that occur during the software development lifecycle. This section, which is normative, corresponds to IEC 601-1-4. [1] This correspondence will assist U.S. manufacturers intending to export to Australia, Canada, the European Union, and Singapore. It will also assist ODE scientific reviewers to evaluate submissions for products developed in accordance with this standard. Manufacturers may chose to follow another standard. However, they would need to demonstrate how that standard corresponds to this guidance document. More detailed information about the development lifecycle is provided in Appendix A, which is informative.

2.1 Lifecycle Models and Development Methodologies.

Manufacturers are responsible for selecting, justifying and following a particular lifecycle model and software development methodology. Many software development models are acceptable. Terminology from model to model and methodology to methodology may vary. A generic lifecycle model is depicted in Figure 2-1. Note that the software development lifecycle is a microcosm of the entire device development lifecycle. It is feasible to intermix lifecycle methodologies between subsystems and subcomponents (i.e., hardware, software, materials, ...).

The Agency looks for four particular characteristics in any lifecycle model:

1) distinct phases with associated work products;

2) feedback of technical and schedule information among the phases;

3) verification and validation activities return to the source of the error, not necessarily the previous

[1] - FDA’s published "Guideline on the General Principles of Process Validation" defines validation as “Establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality attributes.” This definition encompasses both “verification” and “validation” activities as used in this guidance document. This guidance document uses the terms “verification” and “validation” activities as defined in ISO 8402 and used in IEC 601-1-4, which is a common practice for the computer industry.

phase, and look at adjacent and similar areas in the software for associated errors; and

4) development lifecycle and risk management activities are fully integrated.

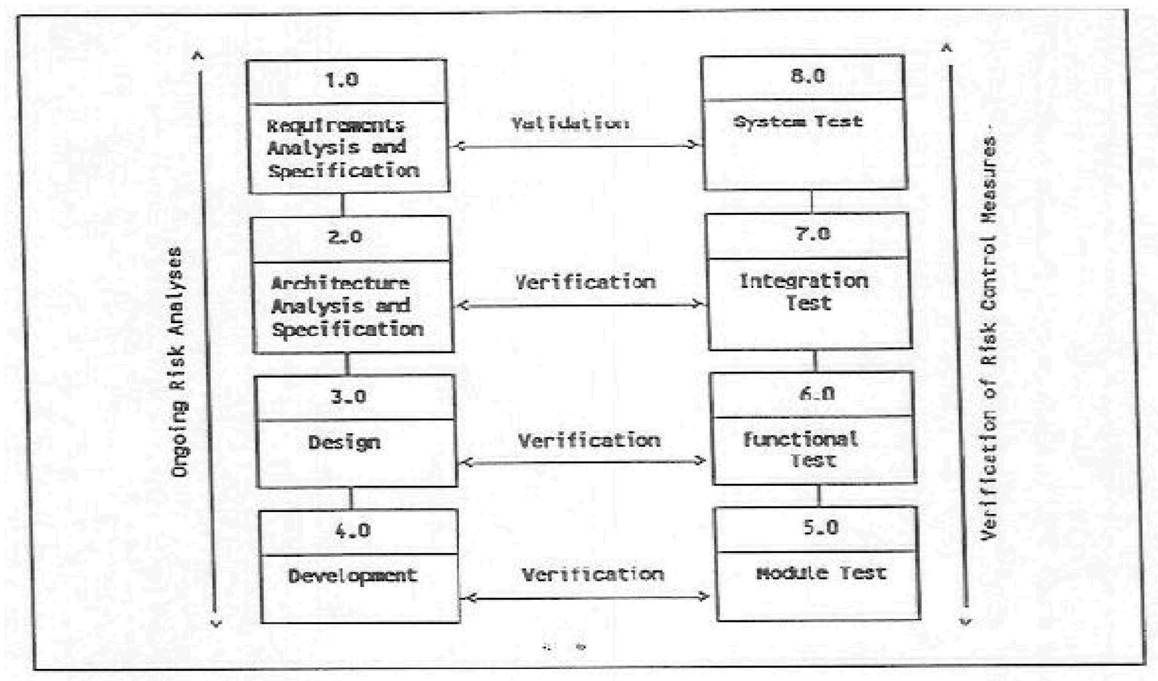

Figure 2-1. Generic Software Development Lifecycle Model. (Adapted from IEC 601-1-4 Figure DDD.1.)

2.2 Requirements Analysis and Specification.

The first phase for any medical device software is to identify and analyze customer or end-user functional and performance requirements. This involves the development of specifications which detail risk-related functions and identify the safety integrity measures necessary to control risks for the medical device software. It is important to define the role of the software in the device at this time, in particular with regard to risk-related functions. Prototypes can be useful in clarifying the accuracy and completeness of requirements. This is an appropriate time to start writing a preliminary operator's manual.

2.3 Architectural Analysis and Specification.

The second phase is architectural decomposition and analysis. During this phase functional and safety requirements are allocated to the various subsystems and subcomponents. The role of software in risk control measures should be defined. Manufacturers are responsible for selecting and justifying architectural considerations such as redundancy, diversity, failure rates and modes, diagnostic coverage, common cause failures, systematic failures, test interval and duration, maintainability, data security and privacy. They should explain how and why these architectural considerations were incorporated; documenting the decision making process. For example, how was redundancy implemented and why; or why was it not incorporated.

2.4 Design and Development.

The logic used in performing the design is captured at all levels. These activities develop detail designs of algorithms which are recorded in a detail design specifications. The target and development hardware platforms, operating systems, tools used (such as compilers, linkers, and automated software engineering tools) are part of this specification. The detail design of interfaces with devices (such as sensors, actuators, and human factors issues (see B.10)) are documented in this specifications. The decision making process (especially the detail designs of interfaces with devices), which is documented in the specification, should explain how and why these design considerations were incorporated. The source, object, and executable code with supporting documentation are written.

2.5 Verification Activities.

Verification activities should evaluate the realization of the safety objectives and verify the correct implementation of functional and safety requirements. Test specifications for the software system and each of its subsystems and subcomponents are developed. The results of verification activities should be documented, analyzed, and interpreted; e.g. a discussion of why observed results were considered acceptable -- not just that they passed.

There are two special concerns in regard to verification activities: 1) the use of off-the-shelf (OTS) software products (see B.6), and 2) encouraging the use of complementary analysis and verification techniques. The use of OTS software products is being encouraged by industry. Solely performing verification on commercial software products is not sufficient for acceptance in safety critical applications. Verification and validation activities should evaluate the safety, reliability, and integrity of the OTS product and its intended use in medical device software, and allow for appropriate safeguards to be designed and developed for the device. See Appendix B.11 for a thorough discussion of issues related to OTS.

The second concern is encouraging the use of multiple complementary analysis and verification techniques, as shown in Figure 2-2. By using multiple techniques, a larger number and different types of errors will be uncovered; thereby enhancing software safety and reliability. (Note: Figure 2-2 is only an example; manufacturer's schemes may differ.)

ANALYSIS	FUNCTIONAL	LOGICAL
Dynamic	Traditional Testing: module subsystem system integration stress regression	Trajectory-based testing Structured Basis testing Branch testing Path testing
Static	Formal Scenario Analysis Code Inspections Cleanroom Analysis HAZOP Analysis	Petri Nets Timing Analysis Testability Analysis Critical Path Analysis Formal Methods, Proofs Modelling Software FTA, FMEA, ...

Figure 2-2. Complementary Analysis and Verification Techniques.

2.6 Validation Activities.

Validation activities demonstrate that the safety requirements have been implemented correctly as specified. During validation the results observed should be documented, analyzed, and interpreted. This is an opportunity for the manufacturer to demonstrate that the device has been adequately validated. (See A.6.)

2.7 Configuration Management and Change Control.

Modifications will occur during the development lifecycle, after a device is fielded, and as a product line matures. Modifications may take the form of requirements changes, design changes, corrections, or enhancements. The extent and nature of the modifications will determine whether: (1) they can be accommodated by configuration management and change control procedures; or (2)the activities of the entire development lifecycle apply. Of primary concern is the adequacy of the management and control of risk resulting from changes to the software. (See B.4.)

2.8 Risk Management.

Comprehensive risk management is a combination of risk analysis and risk control activities which are ongoing throughout the development lifecycle. Several national and international consensus standards, such as those cited in Appendix D.2, can assist manufacturers during this process.

2.8.1 Risk Analysis Activities.

Risk analysis begins with the identification of all potential hazards for a device. A variety of inductive and deductive techniques should be used to perform the risk analysis, as cited in Figure 2-2. In general, different techniques will be used during different phases of the development lifecycle as more becomes known about the end product. The severity of each hazard, should it occur, is then assessed qualitatively. (IEC 601-1-4 defines the categories of severity as: negligible, marginal, critical, and catastrophic.)

A device may have multiple potential hazards associated with it. Likewise, each hazard may have multiple potential causes. All potential causes for each hazard should be identified. The estimated likelihood of each hazard occurring is then assessed, either qualitatively or quantitatively. The methods used to identify hazards and their causes, estimate the likelihood, and categorize severity should be documented.

2.8.1.1 Estimating the Likelihood and Severity of Hazards.

Risk analysis activities should lead to an identifcation of hazards and an associated risk of each hazard. This involves estimating the likelihood that the hazard will arise, estimating that the hazard will cause a mishap, and estimating the severity of the mishap.

The likelihood of a hazard can be estimated in different ways. Some examples are:

1) estimation through known problems associated with a specific technological issues or information through literature;

2) estimation through experience/engineering judgement in using a technology; and

3) estimation through review process when the likelihood is purely an estimate based on knowledge of team and/or technology.

The reason the likelihood of a hazard is usually estimated is that calculating specific probablilities of occurance for a particular hazard is difficult, if not impossible, depending on the circumstances which contribute to the hazard. This estimate should be used to drive the level of risk analysis and control for a particular hazard when information or engineering judgement reasonably shows a particular concern for a hazard [2].

2.8.2 Risk Control Activities.

Risk reduction and mitigation techniques are employed to control the severity of a hazard and/or the likelihood of it occurring. The order of precedence for risk control activities is:

1) reduce the risk by inherent safe design or redesign,

2) reduce the risk by protective measures, and

3) reduce the risk by sufficient warnings.

These activities are often used in conjunction with each other for high risk and/or complex devices.

A determination is made about the appropriateness of the residual risk for each hazard/cause combination. Following this, a determination is made as to whether or not the risk control measures introduced any new hazards. If so, the process is repeated. Referring to Figure 2-3, steps 2 through 7 are repeated for each potential hazard while steps 4 through 6 are repeated for each potential cause. After all potential hazards have been evaluated, a final determination is made about the device safety. Several work products result from the ongoing risk management process. A hazard analysis by itself is not sufficient. Manufacturers should also document:

- what hazard analysis techniques were used;
- what the estimated likelihood of each hazard occurring is and how it was estimated;
- what the estimated severity of each hazard is and how it was categorized;
- what risk reduction and mitigation techniques were implemented and how their effectiveness was assessed; and
- testing and evaluation demonstrating the implementation of the safety features.

---

[2] - Sommerville, Ian; “Software Engineering”; Chapter 2

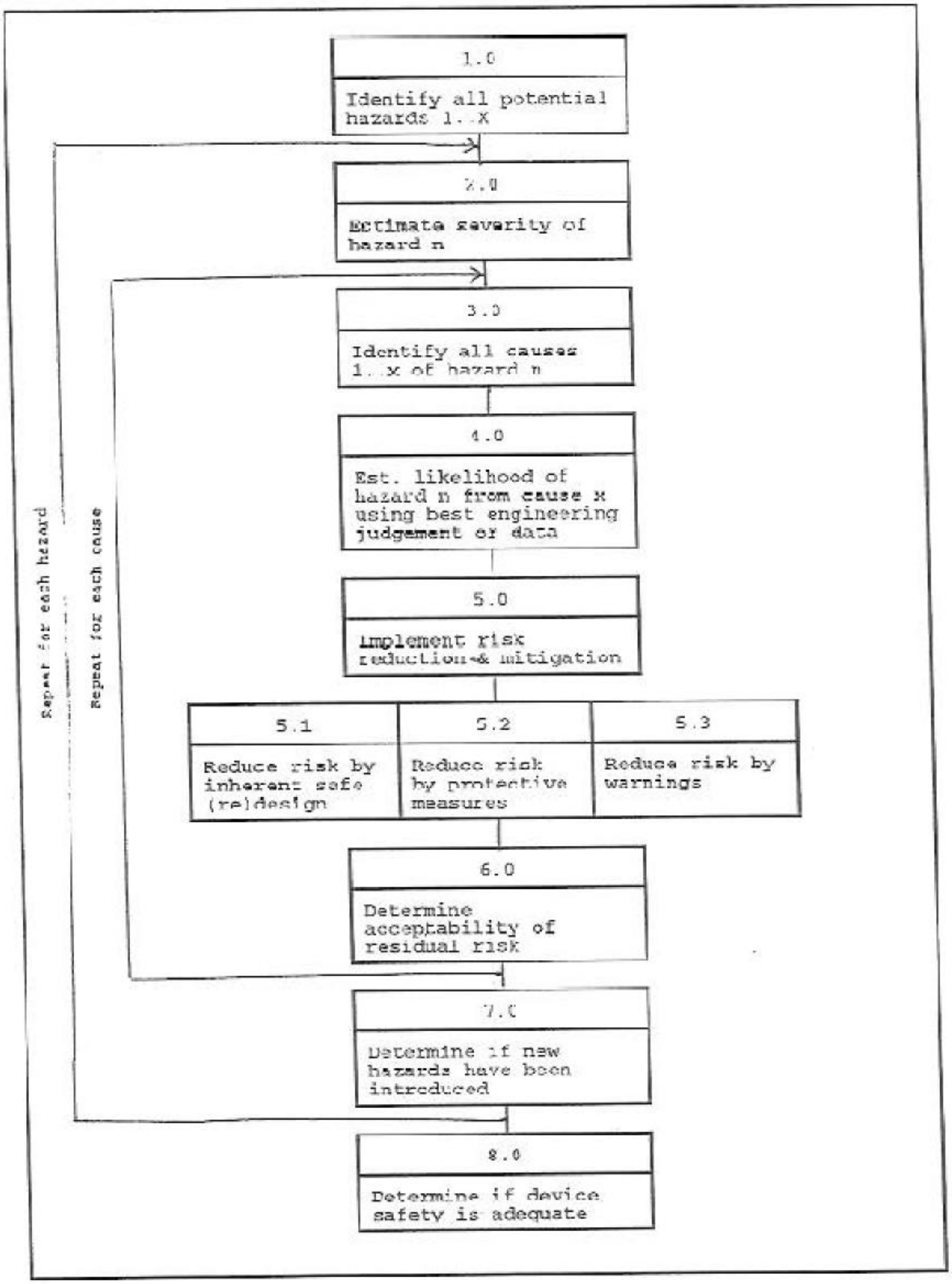

Figure 2-3. Risk Management Process. (Adapted from IEC

601-1-4 Figure CCC.2.)

A standard risk management template, such as that shown in Figure 2-4, can be used to document this information. Sections 2 through 12 are repeated for each hazard, while Sections 5 through 11 are repeated for each cause.

RISK ANALYSIS

1. Hazard analysis technique(s):________________________
____

2. Hazard(s)identified:________________________
____

3. Maximum tolerable risk:________________________
____

4. Severity category (negligible, marginal, critical, catastrophic):________________________
____

4.1 Can a failure be detected before a hazard occurs?____
During what interval?________________________
____

4.2 What techniques have been employed to reduce the severity of the hazard?________________________
____

5. Cause(s) (software, hardware, etc.)________________
____

6. Likelihood using best engineering judgement or available data (incredible, improbable, remote, occasional, probable, frequent):____________________
____

6.1 Does the hazard occur in the absence of a failure, in failure mode only, or in multiple failure mode only?
____
____

6.2 What techniques have been employed to reduce the likelihood of the hazard?________________________
____

7. Estimated residual risk?________________________
____

Figure 2-4. Standard Risk Management Template: Part I - Risk Analysis. (Adapted from IEC 601-1-4 52.201.3.)

Keep in mind that risk analyses should be performed for the device as an entity. Appropriate techniques must be chosen so that hazard analyses for the software, electronics, biomaterials and so forth, can be effectively integrated and analyzed at the device level as well.

RISK CONTROL

8. Minimum safety control requirement___________________
____

9. Implemented safety control___________________________
____

10. Safety integrity (likelihood of a safety-related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time)______________________________
____

11. Verification activities______________________________
____

12. Validation activities________________________________
____

Figure 2-4. Standard Risk Management Template: Part II - Risk Control. (Adapted from IEC 601-1-4 52.201.3.)

SECTION 3. Level of Concern

FDA/CDRH uses the term "level of concern" to mean the severity of risk that a device could permit or inflict (directly or indirectly) on a patient or operator as a result of latent failures, design flaws, or using the medical device software. The extent of the regulatory review process is proportional to the level of concern. Therefore, it is important to clarify the role of software in causing, controlling, and/or mitigating these types of events. Manufacturers should state: 1) the level of concern for the software and the device; and 2) how the level of concern was determined. This section provides suggested evaluation criteria that should be used to establish the level of concern for computer-controlled medical devices; it is normative. This approach should also be used in determining the severity of each hazard identified in the hazard analysis.

3.1 Background.

Risk(s) from potential failures or possible design flaws is a concern during the review of medical device premarket submissions containing software. Inadequate or inappropriate software development lifecycle and risk management activities, inappropriate use of a medical device, and/or operational errors could lead to unsafe or ineffective delivery of energy, drugs, life-supporting or life- sustaining functions, or to incorrect or incomplete information causing a misdiagnosis or selection of the wrong treatment or therapy.

The level of concern for medical device software varies over a continuum. Accordingly, it is essential that the criteria on which to determine the level of concern be straightforward. If level of concern determinations have already been made for specific devices and review criteria are already established by Division management, Division review procedures, and/or by the Office of Device Evaluation, they would take precedence.

Various national and international consensus standards and references (Appendices D and E) classify the severity of risk on an incremental scale. This is helpful for determining the appropriate degree of rigour and the kinds of risk management activities that should be performed. We encourage manufacturers to use the risk identification and control techniques promoted in standards and references.

3.2 Approach Recommended by FDA.

As mentioned in Section 2, the two components of risk estimation are likelihood and severity. The correlation of likelihood and severity is used to determine the acceptability of risk during both: 1) the initial risk analysis; and 2) the assessment of residual risk after risk control measures have been implemented. This correlation yields three risk regions: unacceptable, as low as reasonable practicable (ALARP), and broadly acceptable. To illustrate, a potential risk that occurs frequently but has negligible severity is considered broadly acceptable; while, a risk that occurs occasionally but has a catastrophic severity is unacceptable. (See Figure 3-1.)

Furthermore, it is indicated in IEC 601-1-4 that device specific standards in the IEC 601-2-x series should also be consulted when determining the acceptability of residual risk. Accordingly, FDA/CDRH has developed the concept of level of concern. The approach discussed below explains how to determine into which region a risk falls, with a higher level of concern equating to the ALARP region and a lower level of concern equating to the broadly acceptable region.

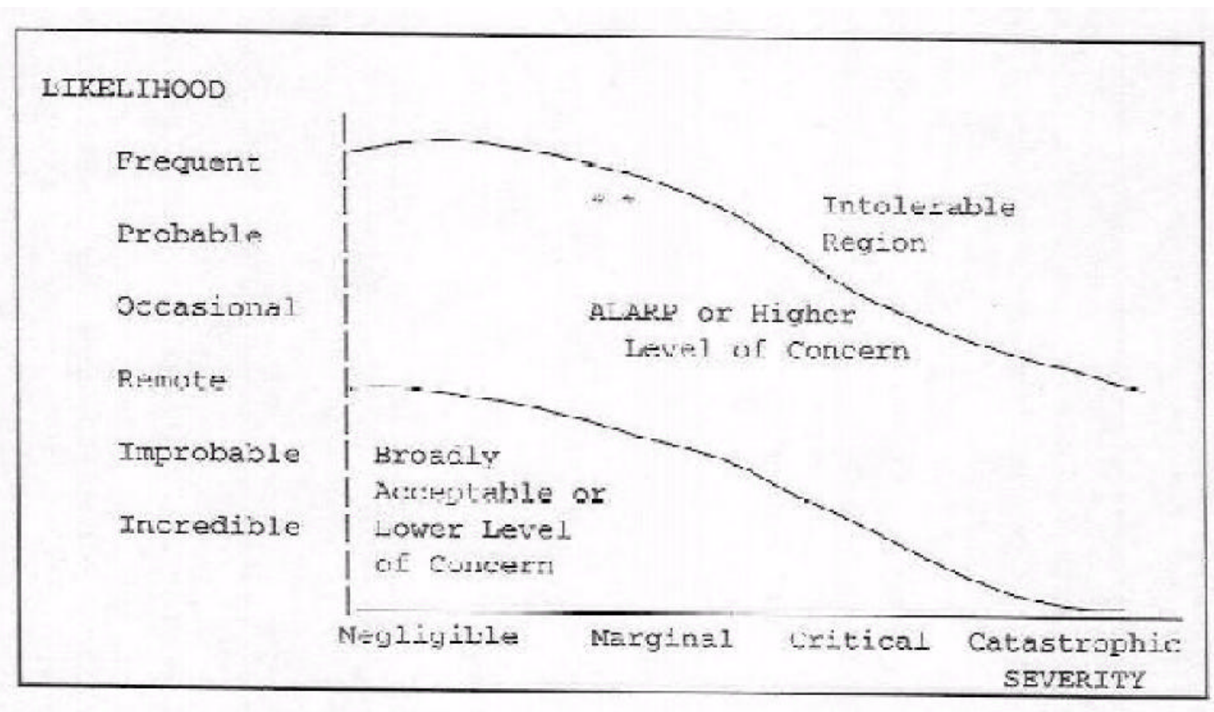

Figure 3-1. Risk Regions. (Adapted from IEC 601-1-4 Figure BBB.1.)

The approach to determining level of concern yields either a higher or a lower level of concern. The purpose of the higher and lower categories is to streamline the review process into two categories that are easy to differentiate and understand by both reviewers and industry. The level of concern determination is being streamlined for review purposes only. The medical device software function and associated risk will dictate the scope of the review.

3.2.1 Definitions.

Definitions associated with the lower and higher level of concern relate to the consequences if the software were to fail:

Lower - The level of concern is lower if latent failures or design flaws (direct or indirect) would not be expected to result in death or serious injury. This corresponds to the broadly acceptable region in IEC 601-1-4 Figure BBB.1. (See Section 3.2.)

Higher - The level of concern is higher if latent failures or design flaws (direct or indirect) could result in death or serious injury. This corresponds to the ALARP region in IEC 601-1-4 Figure BBB.1. (See Section 3.2.)

Serious injury, as defined in the final Medical Device Reporting (MDR) regulation in the Code of Federal Regulations 21 CFR 803.3 (aa), means an injury or illness that:

i. is life threatening,

ii. results in permanent impairment of a body function or permanent damage to a body structure, or

iii. necessitates medical or surgical intervention to preclude permanent impairment of a body function or permanent damage to a body structure.

Permanent means for the purposes of this subpart, irreversible impairment or damage to a body structure or function excluding trivial impairment or damage.

3.2.2 Decision Process.

The level of concern for medical device software may be determined using the following process made up of five key questions which are asked in sequence. If the answer to any one (or more) of them is yes, the software is of higher level concern. Note: the software may have the same or lower level of concern than the device, but not higher. If the answer to any question is no, continue on to the next question. (Refer to Figure 3-2).

1. Does the device software control a life supporting or life sustaining device?

2. Does the device software control the delivery of potentially harmful energy which could result in death or serious injury, such as radiation treatment systems, defibrillators and so forth?

3. Does the device software control treatment delivery, such that an error or malfunction with the delivery could result in death or serious injury?

4. Does the device software provide diagnostic information on which treatment or therapy is based, such that if misapplied it could result in serious injury or death?

   If yes, then the device may be of a "higher" level of concern, especially in cases where diagnostic information would be misleading or inaccurate, or unsuitable licensed biologic products are released. These situations may result in serious injury to the patient through an improper diagnosis of a serious medical condition or improper treatment.

   A device may provide data for which it is unlikely that the clinician will exercise independent judgement. In this case, two scenarios may arise:

   Higher: a system releasing HIV infected blood without user knowledge or treating a patient with potentially harmful drugs/therapy for a particular illness or condition based on incorrect results from a medical device that are difficult or impossible to detect or override; and

   Lower: a system providing incorrect diagnostic information which is easily detected and overridden based on patient demographics, symptoms, and other tests.

   In the latter case, one in which clinical judgement would be exercised to override the information provided by a medical device (even in cases of incorrect data), the device would be considered one of lower level of concern.

5. Does the device software provide vital signs monitoring and alarms for potentially life threatening situations in which intervention is necessary?

In conclusion, reviewers should expect reasonably complete evidence to support the level of concern determination for medical device software. (See Section 4.1.1.)

```
+))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
*                     +))))))))))))))))))),                   *
*                     *   Does the        *                   *
*                     *    device         *                   *
*                     *   software:       *                   *
*                     .))))))))0))))))))))-                   *
*                              *                              *
*                              *                              *
*                     +))))))))2))))))))))),                  *
*                     *       1.0         *                   *
*                     /)))))))))))))))))))1                   *
*                     *Control a life     *                   *
*                     *supporting or      *                   *
*                     *sustaining device? *                   *
*                     .))))))))0))))))))))-                   *
*           yes                *                              *
*    +<))))))))))))))))))))))))1                              *
*    *                +))))))))2))))))))))),                  *
*    *                *       2.0         *                   *
*    *                /)))))))))))))))))))1                   *
*    *                *Control delivery of *                  *
*    *                *harmful energy?    *                   *
*    *                *                   *                   *
*    *                .))))))))0))))))))))-                   *
*    *      yes                *                              *
*    /<))))))))))))))))))))))))1                              *
*    *                +))))))))2)))))))))),                   *
*    *                *       3.0        *                    *
*    *                /))))))))))))))))))1                    *
*    *                *     Control      *                    *
*    *                *    treatment     *                    *
*    *                *    delivery?     *                    *
*    *                .))))))))0)))))))))-                    *
*    *      yes                *                              *
*    /<))))))))))))))))))))))))1                              *
*    *                +))))))))2))))))))))),                  *
*    *                *       4.0         *                   *
*    *                /)))))))))))))))))))1                   *
*    *                *Provide diagnostics *                  *
*    *                *as a basis for     *                   *
*    *                *treatment/therapy? *                   *
*    *                .))))))))0))))))))))-                   *
*    *      yes                *                              *
*    /<))))))))))))))))))))))))1    no                        *
*    *                +))))))))2))))))))))),                  *
*  +)2))))))))))),    *       5.0         *    +)))))))))))))),*
*  * Higher Level  *  /)))))))))))))))))))1    *Lower Level   **
*  *     of        *  *  Perform vital    *    *    of        **
*  *   Concern     *  * signs monitoring  *    *  Concern     **
*  .)0))))))))))))-   .))))))))0))))))))))-    .))))))))))))0)-*
*    *      yes                *                no          *  *
*    .<))))))))))))))))))))))))2)))))))))))))))))))))))))   >-  *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

Figure 3-2. FDA's Approach to Deciding Level of Concern.

SECTION 4. Documentation in a Premarket Submission

Premarket submissions for computer-controlled medical devices should contain documentation consistent with the level of concern, intended use of the device, and type of premarket submission. This section provides a checklist of the items for review (4.1) and discusses how this may be affected by the level of concern (4.2). This section also discusses the differences between the types of premarket submissions (4.3) and the impact of software and system changes (4.4). This section is normative.

Risks, complexity, design issues, technology, and methodology vary widely across the medical device and manufacturer spectrum. Hence, the information presented is generic and may be more or less rigorous than what is necessary for a specific device under review. Reviewers should: 1) use professional discretion in reviewing a computer-controlled medical devices; 2) follow any guidance provided by ODE management, Division management, and/or device-specific guidance; and 3) consult with other reviewers who are knowledgeable about software engineering practices.

Manufacturers are encouraged to contact the corresponding FDA/CDRH ODE division prior to making a submission to obtain any additional pertinent information. This will promote more complete initial submissions. Also, consult Appendix C for examples of boilerplate questions and a review items checklist.

4.1 Review Items.

This subsection identifies items that should be considered when reviewing a computer-controlled medical device. Other issues which may be relevant when reviewing a computer-controlled device are discussed in Appendix B. The review items listed following each subsection of 4.1 represent the high-end of information. Each list is preceded by the standard phrase: "some items to consider, when appropriate, are". This indicates that not all items may apply to all devices or that the list is exhaustive. If the information discussed in this section is not submitted for the device under review, then adequate justification should be submitted in lieu of the data unless it is obvious that the information is not pertinent.

The wording in this section is not intended to imply any specific document structure but instead data needed to make a determination. The manufacturer is free to organize their documentation in any manner needed for their internal operation. Data submittals could include electronic media where appropriate provisions and negotiations with the FDA have been provided.

There are five major components of a premarket submission for a device containing software. They include information about the level of concern, descriptive data about the development and intended operational environment, labeling, development lifecycle activities, and risk management activities. This information is summarized in Figure 4-1.

4.1.1 Level of Concern.

The first component is the level of concern. As mentioned in Section 3, FDA/CDRH uses this term to mean the severity of risk that a device could permit or inflict (directly or indirectly) on a patient or operator as a result of latent failures, design flaws, or using the device. Documentation should be consistent with the level of concern for the device under review. The contribution of software to the hazard shall be clearly documented and explained. The analysis used to arrive at the level of concern should be included. Manufacturers should state the level of concern for each identified hazard, which hardware and software components contribute to the hazard and how the level of concern for each component was determined, using Figure 3-2 as a template.

4.1.2 Descriptive Data.

The second component is descriptive data about the software development environment and intended operational environment. This should include a list of the:

a. hardware platform;

b. operating system;

c. compiler;

d. any simulators, emulators, off-the-shelf-software (OTS) and automated software engineering (ASE) tools used; and

e. concurrent applications, anticipated system load, and indication of single or multi-user mode.

FDA SOFTWARE GUIDANCE	IEC 601-1-4 REFERENCE
4.1.1 Level of Concern each hazard device how it was determined	52.204.3.2 52.204.3.2.3, 52.204.3.2.4
4.1.2 Descriptive Data development environment intended operational environment	 52.207 52.208
4.1.3 Labeling intended use instructions for use known non-hazardous anomalies	 6.8.201 6.8.201
4.1.4 Development Lifecycle Activities requirements specifications architecture analysis design specification test specification verification plan validation plan analysis of validation results configuration management and change control compliance assessment report	52.203 52.206 52.207 52.208 52.208 52.209 52.210 52.210.6 52.201.2, 52.211 52.212
4.1.5 Risk Management risk management plan risk analysis hazard identification method(s) risk likelihood estimation and estimation method(s) severity estimation and categorization method(s) risk control measures evaluation of effectiveness of risk control measures	 52.202 52.204.3 52.204.3.1.8 52.204.3.2 52.204.3.2.3 52.204.4 52.204.6

Figure 4-1. Summary of potential documentation items for a new premarket software submission.

4.1.3 Labeling.

The third component is labeling, which includes a discussion of the intended use and instructions for use. The labeling should be appropriate for the device to ensure that the device can be used safely and for the purposes for which it is intended. Labeling of a medical device should be consistent with the requirements discussed in the FDA/CDRH ODE Blue Book Memo "Device Labeling Guidance #G91-1", dated March 8, 1991, or newer version. Some items to consider, when appropriate, are:

a. Consistency between intended use and software/system requirements;

b. Hazardous operating procedures identified in precautions and proscribed in warnings;

c. Listing of compatible equipment or additional equipment required;

d. Operating and training (e.g. application) manuals;

e. Instruction and qualification checklist for installation;

f. Trouble shooting guide, device configurations, and error message information;

g. Listing of known non-hazardous anomalies; and

h. Current, complete, and understandable instructions for use.

4.1.4 Software Development Lifecycle Activities.

The fourth component represents the documentation or work products from each of the development lifecycle phases and reflects ongoing configuration management and change control procedures. Figure 4-2 represents a general layout of the software development process. Keep in mind that it is the informational content of the work products that is important; i.e. work products do not necessarily represent separate documents.

PHASE	INPUTS	ACTIVITY	VERIFICATION	WORK PRODUCTS
Requirements Analysis and Specification	- system requirements - project, risk management, and development plans - software QA and requirements standards	- define software quality characteristics - generate software requirements - identify safety requirements - begin developing test plans and cases	- review of requirements - assess adequacy and appropriateness of safety - review functional test cases and plans - assess quality attributes	- software requirements specification - safety traceability matrix - requirements traceability matrix - preliminary hazard analysis - functional test cases and plans - preliminary user's manual
Architectural Analysis and Design	- system requirements - software requirements - preliminary hazard analysis - project, risk, and development plans - software QA and architectural / design standards	- allocation of requirements - develop sub-system, component, interface, data structure designs - modular decomposition - design test plans/cases	- evaluate and review architecture and design - assess adequacy and appropriateness of safety - review test cases and plans - assess quality attributes	- software architecture and design documents - verification activities reports, design review documentation - updated traceability matrices and hazards analysis - test cases and plans
Development	- software requirements/ design - project, risk management and development plans - software QA and coding standards	- develop source, object, and executable code with supporting documentation - debug - design test plans/cases	- code walkthroughs and inspections - static and dynamic analyses - unit/module/ subsystem testing - safety/failure analyses - review test, data, cases, and plans - assess quality attributes	- source code and all supporting documentation - verification activities reports - updated traceability matrices and hazard analysis - testing documentation
Test	- test cases and plans - source/executable code - software QA, risk management plan and test completion criteria standards	- unit/module/sub system - integration - performance, stress, functional, structure, fault, safety, system, and beta tests	- review and analysis of testing - review testing and traceability to requirements, functional, performance, and safety - assess quality attributes	- verification activities report - resting documentation - verification and validation report - safety assessment report

Figure 4-2. Relationship between generic development lifecycle phases and work products.

4.1.4.1 Lifecycle Model.

Documentation should be consistent with the software development lifecycle model, software engineering methods and practices. Some items to consider, when appropriate, are:

a. Discussion of software development lifecycle model, methodologies and policies;

b. Discussion of the graphical, symbolic, and notational models that are conventional for the analysis and design method used;

c. Discussion of error logging and tracking;

d. Results of design reviews, code walkthroughs, software audits, and independent verification and validation; and

e. Discussion of configuration management and change control procedures.

4.1.4.2 Specifications.

Specifications resulting from requirements analysis, architecture analyses, and design tradeoff analyses or analysis or alternatives should be developed and maintained for the device and each of its subsystems and subcomponents. Tables, charts, diagrams, and/or calculations should be used to present the information contained in specifications wherever possible; certain information can be conveyed more concisely this way. Documentation should provide a clear and consistent description of system and software requirements. Some items to consider, when appropriate, are:

a. Hardware requirements;

b. Performance, functional, and safety requirements;

c. Algorithms for therapy, diagnosis, monitoring, and interpretation (with citations);

d. Device limitations due to software;

e. Internal software tests and checks;

f. Error and interrupt handling;

g. Timing requirements; and

h. Communication protocols.

4.1.4.3 Verification and Validation Activities.

Verification and validation activities should provide traceability to software and device requirements. The test report should explain what level of test coverage was necessary for the device and how it was achieved. The results of verification and validation activities should be analyzed and interpreted. Some items to consider, when appropriate, are:

a. System level test protocol with pass/fail criteria;

b. Verification and Validation report discussing how the phases and methods used demonstrate that requirements were met;

c. Results and analysis of the following (when appropriate):

- Fault, alarm, and hazard testing;
- Error, range, and boundary value testing;
- Timing analysis;
- Special algorithms;
- Path analysis and branch testing;
- Stress testing;
- Testing of all device options, accessories, and configuration(s);
- Communications testing;
- Memory utilization testing;
- Verification of OTS software;
- Acceptance and beta site testing;
- Estimation of residual defects;
- Regression testing; and
- Test completion criteria, including test case approach and design.

4.1.4.4 Compliance Assessment Report.

In a compliance assessment report, manufacturers should report any national and/or international consensus standards that were used during the software development lifecycle and how compliance to these standards was demonstrated.

4.1.5 Risk Management Activities.

The fifth component represents the documentation or work products resulting from risk management activities. A hazard analysis by itself is not sufficient. Manufacturers should also document: the hazard analysis techniques used; the estimated likelihood of each hazard occurring and how it was estimated; the estimated severity of each hazard and how it was categorized; and risk reduction and mitigation techniques implemented and how their effectiveness was assessed. (IEC 601-1-4 Figure BBB.1 defines the categories of likelihood as being: frequent, probable, occasional, remote, improbable, and incredible.) A standard risk management template, such as that presented in Figure 2-4 can be used to document this information. Hazard analyses should be performed for the device as an entity. Appropriate techniques must be chosen so that hazard analyses for the software, electronics, biomaterials and so forth can be effectively integrated and analyzed at the device level as well.

4.2 Level of Documentation.

The extent and nature of a review and its supporting documentation is proportional to the level of concern for the software. Two levels were identified in Section 3: lower and higher. As stated previously the level of concern is a continuum, not discrete levels. Software of lower level of concern will not involve the same level of review as higher level of concern software. The documentation submitted for lower level of concern software may be more summary in nature, while higher level of concern software should contain more descriptive data and analysis regarding the development lifecycle activities. The following scenarios explain how to use the review criteria established in 4.1 for device software of varying levels of concern.

4.2.1 Lower Level of Concern Software.

A submission for lower level of concern software should include the information identified in 4.1.1 Level of Concern, 4.1.2 Descriptive Data, 4.1.3 Labeling, and 4.1.5 Risk Management Activities. Safety-critical and performance-critical functions should be highlighted and discussed in the requirements traceability matrix and in the test analysis report. (See Figure 4-1.)

Information provided for development lifecycle activities (4.1.4) may be more summary in nature and demonstrate: 1) that an appropriate software lifecycle model was followed; and 2) that an appropriate development environment utilized, for the device under review. (See Figure 4-1.)

Some software development information listed in 4.1 may not be appropriate for a lower level concern device; if so, this should be adequately explained and justified by the manufacturer.

4.2.2 Higher Level of Concern Software.

A submission for higher level of concern software should include the information identified in 4.1.1 Level of Concern, 4.1.2 Descriptive Data, 4.1.3 Labeling, and 4.1.5 Risk Management Activities. Safety-critical and performance-critical functions should be highlighted and discussed in detail. (See Figure 4-1.)

In contrast to lower level of concern software, information provided for development lifecycle activities (4.1.4) for higher level of concern software would be more descriptive and detailed and include more analysis and interpretation of results. In some cases, a more detailed requirements specification for high risk software with major areas of concern and more detailed verification and validation information would be reviewed to provide a proper assessment of software performance, safety, and reliability. This information should demonstrate that an appropriate software lifecycle model was followed and development environment utilized. (See Figure 4-1.)

Refer also to guidance developed for specific devices and established review procedures in the corresponding FDA/CDRH ODE Division. Manufacturers are encouraged to contact the corresponding FDA/CDRH ODE Division prior to submitting data to obtain information about device specific guidance and established review procedures.

4.3 Types of Submissions.

It is not within the scope of this document to discuss differences between a premarket notification or 510(k), an investigational device exemption (IDE), and a premarket approval application (PMA). However, there are differences that should be noted and taken into consideration when reviewing software documentation for a device.

4.3.1 Premarket Notification - 510(k).

Because a 510(k) is intended to demonstrate substantial equivalence, performance and safety should be assessed and compared to the device for which substantial equivalence is claimed. When a new intended use or new technology is involved, then documentation should be reviewed in order to determine if:

1) new software questions are raised regarding safety or effectiveness; or 2) the data and information provided demonstrate that there are no new software issues of safety and effectiveness. See Appendix B regarding the need for clinical data for a new indication, technology that differs from the predicate device or new algorithm for therapy, diagnosis, or monitoring.

4.3.2 Investigational Device Exemption - IDE.

An IDE is submitted for significant risk devices. This suggests that the software review for a device in an IDE should be consistent with a device of higher level of concern. Because some IDEs involve feasibility studies and not a definitive clinical trial, it may not be possible for the manufacturer to submit complete documentation for a final product as listed in 4.1. However, the information submitted should be consistent with the device configuration used in the feasibility study and should provide sufficient assurance that the software will perform safely and reliably. Manufacturers planning such studies should contact their corresponding FDA/CDRH ODE reviewer and/or Division management prior to submitting an IDE for a feasibility study to discuss/review criteria for an acceptable level of documentation and testing.

Software documentation for an IDE which is beyond the feasibility study phase should remain consistent with the checklist provided in 4.1. It should reflect the configuration of the device to be used in the definitive clinical trial. Because an IDE is for a device in its developmental stages, configuration management, error logging, change control, and software maintenance are vital. Information associated with these activities should be reviewed carefully.

4.3.3 Premarket Approval Application - PMA.

In a PMA a manufacturer is demonstrating the safety and effectiveness of a device. Most PMA devices are class III because they were either not available prior to the device amendments or the risks associated with them are consistent with the controls required for a class III device. Because of the potential higher risk of a class III device, the majority of devices reviewed in a PMA submission will be of a higher level of concern. Many of these devices will also have been submitted under an IDE when undergoing a clinical study. Software documentation submitted in a PMA should include all of the information discussed in 4.1.

4.4 Software Changes and Modifications.

Changes to a device could include new functionality, corrections, and/or migration to new technology. Documentation for changes to a device builds upon that originally submitted. (See Figure 4-3.) The documentation in a premarket submission should reflect the current version of the software and revision history since many minor changes and corrections that do not affect safety and effectiveness occur during a product's lifecycle. The information submitted builds upon that which was submitted in the original document. It should include a description of the changes, what was changed, why it was changed and how the changes affect safety and reliability. The design, functionality, intended use, operation, performance, and safety and reliability features of the new and predicate device should be compared. Traceability between the development lifecycle and risk management activities for the new and predicate device should be demonstrated. A revision history log should be maintained which documents chronologically changes made to the device and its associated lifecycle documentation.

FDA SOFTWARE GUIDANCE	IEC 601-1-4 REFERENCE
4.4 Description of Changes what changed why it was changed impact on safety and reliability	 52.211 52.201.2 52.201.3
4.4 Comparison of New and Predicate Device: design functionality intended use operation performance safety and reliability	
4.4 Traceability to Previous development lifecycle & risk management activities	52.201.2 52.210 52.211 52.212
4.4 Software Revision History affected development lifecycle/risk management: specifications plans procedures	52.201.2

Figure 4-3. Summary of potential documentation items for changes in a premarket software submission.

Deciding when to submit a new premarket submission because of changes made to the software or system of a medical device is another question. For further information on this subject consult the latest version of FDA/CDRH publication "When to Submit a 510(k) for a Change to an Existing Device." (See Appendix E.6.)

4.5 Summary.

This section provided a checklist of information to review in premarket submissions. Depending on the level of concern, the information may be more summary in nature for a lower level of concern device or more descriptive and detailed in nature for a higher level of concern device. This is also true when modifications, enhancements, and upgrades are made to software. Regardless of the type of submission being reviewed, the information submitted should demonstrate:

- How thoroughly the manufacturer analyzed the safety of hazardous functions and implemented safety requirements in the system and software;

- How well the manufacturer established the appropriateness of functions, algorithms, and knowledge on which the system is based;

- How well the manufacturer assessed the device reliability;

- Whether appropriate technology is used to mitigate and/or control hazards, promoting an inherently safe system; and

- Whether the manufacturer has provided adequate software documentation to make a final recommendation about the development, performance, reliability, and safety of a device.

APPENDIX A. Software Development Lifecycle Activities.

This appendix, which is informative, provides more detailed information about the software development lifecycle. The references in Appendix D and the standards in Appendix E have been provided for consultation when:

- developing a framework for software development;
- providing software verification and validation oversight or independent evaluations [3];
- developing and implementing risk management activities;
- conducting software tests;
- performing software documentation reviews; or
- developing, analyzing, and evaluating software products.

It is customary to write a software development plan when starting a software development project. A software development plan should identify key activities and objectives for the project. This may include:

- input and output criteria for each development phase;
- verification and validation activities;
- risk assessment and mitigation;
- types of testing and test completion criteria;
- quality assurance;
- error logging;

[3] - FDA's published "Guideline on the General Principles of Process Validation" defines validation as "Establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality attributes." This definition encompasses both "verification" and "validation" activities as used in this guidance document. This guidance document uses the terms "verification" and "validation" activities as defined in ISO 8402 and used in IEC 601-1-4, which is a common practice for the computer industry.

- tracking and correction;

- configuration management;

- maintenance; and

- management and personnel requirements and responsibilities.

The plan may also outline the life-cycle model, schedule, staffing requirements, project standards, tool usage, testing requirements, and supporting documentation requirements. It is useful at this time to develop a system definition of some sort that clearly describes the problems, goals, constraints, functions, solutions, and acceptance of what is to be developed prior to putting together an overall software development project plan. Of course, the process definition is dependent on the types of software and risk management activities that are associated with a particular product line.

A.1 Lifecycle Models and Development Methodologies.

It is difficult to assign a right and wrong methodology to the set of activities which produce software. Hence, the purpose of this document is not to specify an exact software lifecycle model, but to communicate the types of characteristics it should encompass.

There are a variety of process models, such as: waterfall, spiral, evolutionary, incremental, top-down functional decomposition (or stepwise refinement), and formal transformation. Software products can be produced by any of these models. However, when risk is involved, as in many medical devices, the spiral or incremental model may be more appropriate since it includes integrated risk management activities and phased feedback processes like the waterfall model. This is not to say that the spiral or waterfall models must be followed.

Because of the risks involved with many medical devices, rapid prototyping used to create a quick executable version or evolutionary development based on laws and principles and not on development phases with feedback, may not be appropriate. For this reason, a phased development lifecycle with feedback, including integrated risk management, is more effective. However, an evolutionary development may be acceptable in a cycle of the spiral model in which high system, user interface, or performance risks are encountered. The response to such a case may be to suspend, temporarily, the specification of the overall product for the cycle and to plan for and to develop the next level of prototyping aimed at resolving the high risk issues.

Generically speaking, a lifecycle model should include a requirements analysis and specification phase, a design phase, a development or implementation phase, verification and validation phases, and a maintenance phase. The bottom line here is that a software development lifecycle be understandable, thoroughly documented, results oriented, auditable, traceable and promote appropriate feedback.

The selection of a software development lifecycle model and software development methodology depends on a variety of factors. Likewise, new models and methodologies are continually evolving with technology. Because of this, it is not practical or logical in the regulatory environment to mandate a particular model or methodology; as long as it encompasses general software process characteristics as previously discussed. Hence, manufacturers are responsible for making this selection and justifying it. Manufacturers should choose a development lifecycle and methodology that are appropriate for their product/product line, corporate culture, development environment, and knowledge base. (Figure A-1 illustrates a generic development lifecycle model.)

A.2 Requirements Analysis and Specification.

The first generic phase in any development lifecycle model is to identify and analyze customer or end-user functional and performance requirements, as mentioned in Section 2.2. During this phase, the functions to be performed, controlled, or monitored by the software are documented in a software requirements specifications document that is complete, consistent, and traceable. This information is derived from the overall system requirements specification and other project and management plans. Each software requirement should be verifiable, i.e. traceable throughout development, testable, complete, understood, and consistent. The software requirements specifications generated during this phase should serve as the basis for the software design, and functional test plans and test cases.

Software safety requirements are derived from the preliminary hazard analysis and ongoing risk management activities, as requirements are updated throughout the lifecycle process. During this phase software quality characteristics (acceptance criteria), such as human factors, functional characteristics, response times, output, safety requirements, etc, are defined. This is also an appropriate time to start writing a preliminary operator's manual.

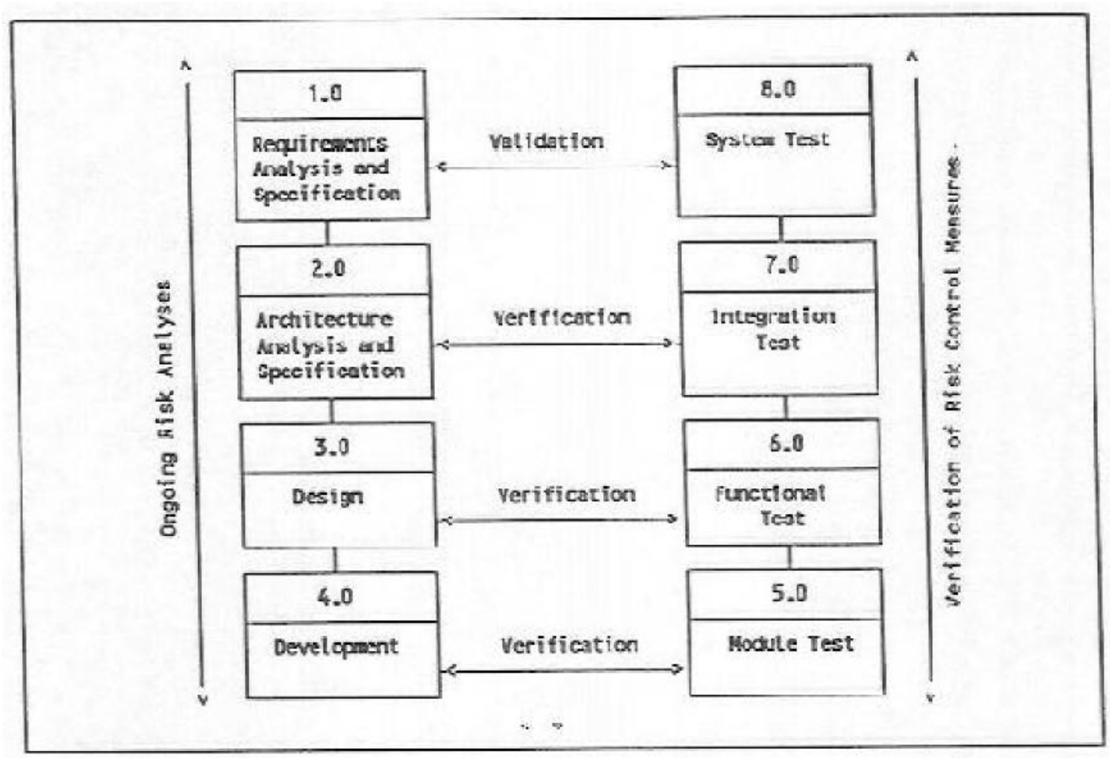

Figure A-1. Generic Software Development Lifecycle Model. (Adapted from IEC 601-1-4 Figure DDD.1.)

A.3 Architectural Analysis and Specification.

The second generic phase in any development lifecycle model is architectural analysis. During this phase functional and safety requirements are allocated to hardware, software, and communications components. Tradeoff studies are performed to determine the most efficient and cost effective allocation of requirements. Interfaces between these components are defined. The analysis performed in this phase is done in an iterative manner as the architecture is continually refined at lower levels of detail. The key decisions made during this phase include the selection of hardware platforms, operating systems, and compilers.

A.4 Design and Development.

The third generic phase in any development lifecycle model is design. During the design phase, analyses are conducted to identify the most efficient way to logically implement the requirements assigned to the software. In the case of a database, both a logical and a physical design will be developed. The goal of design analysis is to optimize the structure, size, and overall performance of the software.

During development, the software requirements expressed in the software requirements specification, architectural considerations, and design are translated into source code. During this phase, appropriate coding standards should be utilized. Code should be well commented with appropriate headers, inline comments, and supporting documentation. Code walkthroughs and inspections should be performed. The source code is compiled to remove syntax errors. Unit testing is conducted to exercise and test the program logic, the control structures, the boundary conditions, computations, comparisons, and control flow. Detailed design diagrams are prepared during this phase. Appropriate corrections are made to the source code and supporting documentation following unit testing. Integration test plans can be generated after design, but before development. It is also valuable to have an independent person or organization generate the test plans.

A.5 Verification Activities.

The fourth generic phase in any development lifecycle model is verification. It is common to hear "verification and validation" spoken together, as if it were one activity. But they are in fact very different. Verification activities, as stated earlier in section 2.5, should evaluate the realization of the safety objectives and verify the correct implementation of functional and safety requirements specification. Validation activities are discussed in Section A.6. Verification activities should determine that the outputs of software development are traceable to and satisfy the requirements established at the beginning of each phase. Although some consider this a "testing" activity, verification also includes analysis, review, inspection, and audit activities.

A preliminary verification plan, which outlines the purposes and activities of verification, can be initiated prior to implementing the software lifecycle process. More details will be included in the plan once the requirements and design phases begin.

Activities and documentation of verification may vary depending on the specific software development phase. There are times when a review and analysis is conducted and documented, and other times when test plans are written and implemented, and results recorded and analyzed. Of course this is dependent on the specific phase and project in question.

A.5.1 Verification of Requirements Analysis and Specification.

Verification should be viewed as an assurance activity for each phase of software development, not just as testing. Software requirements specifications are reviewed for completeness, consistency, effectiveness, testability, traceability, and safety. This verifies that the requirements analysis and specification phase produces appropriate documentation and attributes. Functional test plans and test cases are developed and are traceable to requirements specifications. The traceability matrix is initiated. Hazards identified in the preliminary hazard and safety analyses are addressed with appropriate software and system safety requirements specifications.

A.5.2 Verification of Architectural Analysis and Specification.

During the architectural analysis and specification phase, the architecture of the system and design specifications are developed. The software requirements are, if appropriate, broken down into modules (modularization criteria). Safety requirements are incorporated into an appropriate design architecture. Graphical, symbolic, and notational models that are conventional for the analysis and design method used. Verification of these activities would include evaluating:

- the design specification for consistency, completeness, safety and testability;

- traceability to software, system, and safety requirements;

- design interface analyses; and

- design review documentation.

A.5.3 Verification of Design and Development.

The ultimate goal of the design and development phase is to develop well commented, documented, modular, and structured source code using a programming language that is suitable for the particular application. Utilizing appropriate coding techniques, efficient memory utilization, efficient timing, single entries and exits, and data encapsulation are also goals of the design and development phase. Verification of this phase includes code walkthroughs and inspections, evaluation of completeness, testability, and coding practices, debugging, traceability analysis, and static analysis tools used to assess structural characteristics. Code inspections consist of going through the source code line by line and examining each statement to verify completeness and accuracy. Walkthroughs consist of using data inputs and their associated known outputs to "walk through" the source code to verify that the known results are obtained. Code inspections and walkthroughs are typically performed in teams which track and document all errors found in the code and supporting documentation prior to implementing any corrective action. The verification activities associated with this phase should be documented in order to assess code reviews activities, inspections and analyses.

A.5.4 Verification Testing.

Much of the testing aspect of software is developed in the initial requirements phase where preliminary plans for software testing are initially identified. Throughout the development process, the test plans are modified to account for the software and system requirements, design constraints and considerations, and implementation strategies, and their associated process outputs. Verification testing can be viewed as software testing as it relates to the unit, module, sub-system and integration aspects of software. These test plans are a part of the documentation of the verification aspect of testing, and should be described as part of the software verification plan.

Testing which is designed and written to assess the software performance at various stages of development should be reviewed for accuracy, completeness, and traceability prior to implementing the test plans. Once implemented and complete, the results (data, not just a pass/fail notation) as well as a report and appropriate analyses should also be well documented to assess the test completion criteria prior to final system and acceptance testing. This includes all aspects of testing and analysis, which may include white-box (logic driven) and black-box (data driven) testing, as well as static (no execution) and dynamic (execution) analyses. Verification test plans should include test completion criteria and also include:

- test cases that take into account functional testing (expected normal inputs and outputs);

- boundary value testing (across boundaries such as, data set sizes, ranges, etc.);

- stress testing (intentionally try to break or fool system);

- performance testing (timing and throughput); and

- structure testing (traverse data and logic paths).

All tests should cover special cases in order to track and document software performance in case of invalid or out of range inputs, including various sizes of data sets, and how the system performs error handling and safety functions. This should be traceable, throughout testing, to the software development process and documentation. The test plans should include the testing and expected results, methods and analyses to be used, as well as tools to use for performing tests and measurements.

It should be noted here that at any point during software development assurance activities may uncover defects, inconsistencies, anomalies, and errors that should be documented and traced back to the appropriate source, whether the test plan, code, design or requirements and related documentation, following appropriate change control and configuration management procedures. This documentation should be maintained and tracked as part of the documentation of the software lifecycle. When changes are incorporated, appropriate regression analysis and testing should occur through the software lifecycle process. This information should be analyzed and applied as "lessons learned" to future software projects.

A.6 Validation Activities.

The final generic phase in any development lifecycle model is system validation. Validation is designed to assure that the right product was built; that is to say that it meets stated requirements. Of course, validation also involves other phases of testing as previously mentioned. A test failure during this level of testing is only a symptom of an underlying problem which would need to be traced through the software development process. Validation usually consists of functional, system testing, and acceptance testing. Functional testing is associated with finding discrepancies between the software and its external specification, typically from the end-user's perspective. System validation can include tests for a high volume of data, heavy loads or stresses, human factors, security, performance, configuration compatibility (hardware and software), fault testing (recovery, detection, avoidance, and tolerance), user documentation, implementation of safety requirements, installation and serviceability. Acceptance testing is usually associated with customer acceptance testing and beta site testing, but may overlap with functional and system testing requirements.

Typically beta site testing is performed by the user (or users) or organization in its intended environment to see if the system meets the user's requirements and to find potential weaknesses. It is testing of the software in a clinical site for trade acceptance or as a formal part of validation. Beta site testing may involve treatment of human subjects or it may involve a testing procedure done in parallel with and compared to a currently used competitive product that is in routine clinical use. Beta site testing involving the treatment of a human subject requires IRB concurrence and the submission of an IDE to FDA if the study is judged to be of significant risk.

A final verification and validation report should be generated which documents all verification and validation activities, results, and analyses. As previously discussed, when changes are made during software development, appropriate regression analysis and testing should occur throughout the software lifecycle phases to access the impact of the change.

Complexity also plays an important role as well since more medical devices are becoming software controlled; for example, a radiation therapy treatment planning system. So, it is reasonable to expect a complex system to undergo a more in depth software review. Complexity could mean a combination of one or more of the following: multiple sub-systems, very large programs, multiple processors, complex architecture and design, new technology, etc. The major aspect of a review for such a device would consist of evaluating software lifecycle documentation.

A.7 Configuration Management and Change Control.

Configuration management and change control is an ongoing activity throughout the development lifecycle and the operation and maintenance phase. Configuration management activities include tracking and controlling all work products associated with the software. This includes: requirements specifications, design documentation, source code, object code, test plans and procedures.
A configuration management plan typically consists of defining what is to be managed, how it is to be managed, and who is responsible, i.e.:

- policies for establishing the baseline;
- policies for suggesting, approving, implementing, and recording proposed changes;
- policies for maintaining and identifying versions;
- records describing the configuration management process and individual responsibilities;
- a description of the automated tools used (if any); and
- a definition of the overall tracking system.

During the development lifecycle various project milestones result in formal acceptance of work products, such as the project plan, test plan, user's manual, architectural analysis and design, verification and validation test plan, and source code. At this time these work products are placed under configuration control. Subsequent modifications occur through the configuration management process. Items should be clearly marked to identify the current or correct version of the software and its associated documentation. Of course, this means that software process outputs, such as requirements and specifications, design, source code and various levels of testing are all marked and traceable by version identification.

Change is inevitable. The process of correcting bugs, making improvements and enhancements, and upgrading requirements and specifications is ongoing. There should be a documented process for: (1) establishing the baseline, (2) identifying the change through a change request procedure, (3) undergoing appropriate review and analysis, and (4) identifying appropriate means to updating appropriate documentation and testing activities when necessary. Each change should be subject to regression analysis in order to assess the impact of the change on the software and system requirements and design, source code, and testing activities. This allows for tracking and controlling the revision history of a particular software product. It also ensures that the software development process and evolution is auditable for assessing software traceability, reliability, and safety.

A.8 Independent Verification and Validation.

Verification and validation is a systematic process of lifecycle activities: analysis, evaluation, assurance, and testing of the software and its supporting documentation. To assure that a system is appropriate, reliable, and safe, these activities should involve "outsiders" who have not developed the requirements, design, code or test plans. In an ideal world, independent verification and validation would involve outside third party review, evaluation, and testing; but this may not always be feasible. It may be appropriate to incorporate third party review within a company or organizational unit that is independent of the personnel who developed the software product and documentation. However, the level of criticality should also help determine this as well. For a higher risk device, independent third party review may be desirable. For a small company, third party review may be an even more important consideration since so few personnel have been involved in the software development. Independent analysis may be necessary to uncover design flaws and bugs not apparent to those intimately involved in the project. Verification and validation plans should incorporate third party review, analysis, testing, and auditing to ensure that the lifecycle activities and process outputs are adequate and appropriate, and that the software is safe and reliable.

The following sections from IEC 601-1-4 provide further clarification on the degree of acceptable independence:

> 52.210.3 "The leader of the team carrying out the VALIDATION shall be independent of the design team;"
>
> 52.210.4 "All professional relationships of the members of the VALIDATION team with members of the design team shall be documented in the RISK MANAGEMENT FILE;" and
>
> 52.210.5 "No member of a design team shall validate his own design."

Figure A-2 summarizes the relationship between the generic development lifecycle phases and their associated work products. Not all phases and work products apply to all devices, nor should these phases and work products be considered exhaustive.

PHASE	INPUTS	ACTIVITY	VERIFICATION	WORK PRODUCTS
Requirements Analysis and Specification	- system requirements - project, risk management, and development plans - software QA and requirements standards	- define software quality characteristics - generate software requirements - identify safety requirements - begin developing test plans and cases	- review of requirements - assess adequacy and appropriateness of safety - review functional test cases and plans - assess quality attributes	- software requirements specification - safety traceability matrix - requirements traceability matrix - preliminary hazard analysis - functional test cases and plans - preliminary user's manual
Architectural Analysis and Design	- system requirements - software requirements - preliminary hazard analysis - project, risk, and development plans - software QA and architectural / design standards	- allocation of requirements - develop sub-system, component, interface, data structure designs - modular decomposition - design test plans/cases	- evaluate and review architecture and design - assess adequacy and appropriateness of safety - review test cases and plans - assess quality attributes	- software architecture and design documents - verification activities reports, design review documentation - updated traceability matrices and hazards analysis - test cases and plans
Development	- software requirements/ design - project, risk management and development plans - software QA and coding standards	- develop source, object, and executable code with supporting documentation - debug - design test plans/cases	- code walkthroughs and inspections - static and dynamic analyses - unit/module/ subsystem testing - safety/failure analyses - review test, data, cases, and plans - assess quality attributes	- source code and all supporting documentation - verification activities reports - updated traceability matrices and hazard analysis - testing documentation
Test	- test cases and plans - source/executable code - software QA, risk management plan and test completion criteria standards	- unit/module/sub system - integration - performance, stress, functional, structure, fault, safety, system, and beta tests	- review and analysis of testing - review testing and traceability to requirements, functional, performance, and safety - assess quality attributes	- verification activities report - resting documentation - verification and validation report - safety assessment report

Figure A-2. Relationship between generic development lifecycle phases and work products.

APPENDIX B. Technological Issues and Special Topics

This appendix, which is informative, provides a brief discussion of a variety of special topics that relate to software and premarket submissions. These topics are mentioned to give reviewers a "heads-up" to alert them that the software may incorporate these features and may need special attention or further research. For a more comprehensive discussion of individual topics, consult the references in Appendices D and E.

B.1 Artificial Intelligence, Expert Systems, and Neural Networks.

Artificial intelligence (AI) is a field of research in computer science which studies methods and techniques by which computational machinery may exhibit behavior and responses similar to those exhibited by humans and other biological organisms. Two areas where concepts derived from artificial intelligence research have been applied to problem solving tasks are expert systems and artificial neural networks.

Expert systems attempt to model very specific areas of human knowledge or expertise by distilling the experience of human experts into a set of algorithms which can be executed by software. The expert system often consists of a knowledge base (consisting of rules, heuristics, or relationships between objects or data) and an inference engine which manipulates the knowledge according to selected criteria. Expert systems use these rules or heuristics on facts input into the system to solve problems in a narrow, well-defined domain or area. Typical applications for expert systems include circuit analysis and design, fault detection and diagnosis, automated finance assessment and loan processing, and medical diagnosis and therapy recommendation.

An artificial neural network (ANN) is a data processing architecture that is modelled on principles exhibited in natural or biological neural networks. Artificial neural networks are often used to represent or process nonlinear functions applied to large data sets. Artificial neural network engines can be implemented in software, hardware (using parallel processing architectures) or a combination of both. Artificial neural networks are well-suited for detecting trends or patterns in data, and are typically used for speech and natural language processing, machine vision and image recognition, financial trend forecasting, and automated medical image processing. Artificial neural networks are represented symbolically as an interconnected network of nodes arranged in a specific topology or configuration. Links between nodes represent dependencies between nodes and have weights associated with each link representing the strengths of the dependencies. Artificial neural networks typically have an input layer, hidden or processing layers, and an output layer. The links between nodes, and potentially the topology of the network itself, are adjusted for specific tasks by training of the network, which involves exposing the network to representative data sets to be processed. Output from the network are compared to desired results and corresponding adjustments are made to reduce any discrepancies between the desired output and the actual output. The field of artificial neural networks is a rapidly expanding one, and many artificial neural network models, learning methods, topologies, and training regimens currently exist with others being created constantly.

Expert systems and artificial neural networks are relatively new technologies which are increasingly being incorporated into medical devices. However, they pose special challenges regarding the verification and validation of the core processing architectures: the knowledge base and inference engine for expert systems, and the neural net engine for artificial neural network systems.

The knowledge base of expert systems needs to be verified for accuracy of information and of the relationships between data objects or object classes. The heuristics and rules governing the inference engine need to be analyzed to ensure that there are no logical or common-sense contradictions or paradoxes that exist or that are possible by the system in operation. Any output or determination produced by an expert system should also be accompanied by the reasoning path followed by the software to reach its conclusions. Such information should serve to allow the user to determine if the reasoning path followed by the expert system is sound, or if other valid reasoning paths were not appropriately explored.

Artificial neural networks are, by their very nature, difficult (if not impossible) to qualify using traditional software engineering methodology. The strength of artificial neural networks, the ability of the network to “learn” by example and self-adjust its internal parameters or configuration, is what makes artificial neural network engines problematic. The performance and behavior of artificial neural networks are determined by selective exposure to training sets and its environment, not by strict specifications. In some cases, artificial neural networks can behave in a non-deterministic manner (that is, the same input may produce different outputs at different times). Traditional software engineering methodologies are designed and intended for deterministic software implementations.

The design, assumptions, learning method, and training set data for an artificial neural network need to be evaluated for appropriateness and correctness. The network designers need to justify and explain the choices made for the artificial neural network model, topology, and training sets, as well as explain and justify the data set class that the artificial neural network is intended to analyze or process. The designers need to describe how overfitting or overtraining of the network was avoided, i.e., when it was decided that the network was “trained” sufficiently to enable appropriate performance before the network begins to extract irrelevant details from the data from overexposure to example sets. When examining the training set presented to the neural network, it is important to ensure that the features to be extracted (such as a specific pattern to be detected) remain the common element within the training set data. Once training has been completed, additional data sets should be processed through the network to ensure that the performance remains as expected and relevant data is extracted appropriately. Tests should be performed to ensure that the network was not trained to detect a particular peculiarity of the training set instead of the intended features. Raw data processed by the system should be presented to the user for comparison with the output from the system.

B.2 Automatic Code Generators.

Some computer assisted software engineering (CASE) tools (see B.3) include automatic code generators. Software from automatic code generators must be verified and validated the same way as any other software. This may include, where appropriate, requirements traceability and code walkthroughs.

B.3 Computer Assisted Software Engineering (CASE) Tools.

Computer assisted software engineering (CASE) tools are often used to automate or assist in software development and increase productivity. There are many different tools and types of tools available. At present there is not a single tool or suite of tools from a single vendor which covers all phases of the development lifecycle. This raises concerns about the accuracy of outputs from CASE tools; particularly when developers work in a non-integrated or multi-vendor tool environment.

B.4 Changes and Modifications.

Changes and modifications will occur during the development lifecycle, after a device is fielded, and as a product line matures. Modifications may take the form of requirements changes, design changes, corrections, or enhancements. The extent and nature of the modifications will determine whether:

(1) they can be accommodated by configuration management and change control procedures; or (2) the requirements of the entire development lifecycle apply. Of primary concern is the effect of the modifications on risk analysis and control measures.

Examples of changes include:

1) New hardware platform. This could be migrating to a newer version of the same architecture family, such as i386 to i486, or changing architectures such as from i486 to a workstation.

2) New operating system. This could be migrating to a newer version of the same operating system or changing operating systems.

3) New compiler. This could be migrating to a newer version of the same compiler or changing compilers.

4) New functionality. This includes new features and capabilities that are provided for the end-user.

5) Design enhancements and corrections. This includes changes to the internal software design that may or may not be visible to the end-user. These changes are undertaken to improve software performance, safety, and reliability.

The extent and nature of the changes and modifications will also determine whether a new 510(k) submission is required or information about the changes and their effect on software safety and reliability may be submitted under the "Add to File" provision. Refer to the latest version of FDA/CDRH publication "Deciding When to Submit a 510(k) for a Change to an Existing Device." Changes to an investigational device or PMA device should be handled consistently with those types of devices and submissions. (See Appendix E.6.)

B.5 Clinical Data.

When new algorithms are employed, whether for treatment, diagnosis, or monitoring clinical data may be necessary. This does not imply that every new algorithm needs to be clinically tested, especially since new algorithms may be developed for issues that do not relate to the inherent risk of a device, such as a new communication protocol for an internal printer. Manufacturers, however, should be aware that utilizing new algorithms for various aspects of treatment, diagnosis, interpretation, monitoring, etc. may need to be clinically validated by appropriate clinical trials that yield relevant clinical data and results. The reviewing division within CDRH should also be contacted as early as possible for proper guidance and requirements. Determinations of whether a particular device would be one of significant risk and require an IDE should be made by the reviewing division and/or an institutional review board (IRB). Regardless of whether an IDE is submitted, adequate informed consent should be made available to the patient and the study should be approved by an IRB.

B.6 Closed Loop and Target Control.

Closed loop systems typically include patient feedback, while target control typically "estimates" patient response. In either case, control of a device is based on 'real' or 'estimated' patient data. Typically, these types of devices have required clinical data to support the algorithms on which they are based. In either case, the design and architecture of the system (including software) should allow for partitioning of the system so that complexity is reduced, and safety and testability are maximized. Safety is a critical issue since the clinician is removed from direct control of the device. Adequate risk assessment and mitigation activities should be performed during the software lifecycle process, and failure analyses techniques should include assessing multiple event failures. Some single event failures may not pose a direct safety hazard, however, this may change when multiple event failures occur concurrently or in a particular sequence. Even if a legally marketed device is incorporated into a closed loop or target control system, requalification of the device in order to assess it’s appropriateness for incorporation into one of these kind of systems may be necessary, especially since the behavior in a closed loop system may alter the way a device typically functions. Due to the nature of using a device to either monitor or control therapy in a closed loop system, qualification of the device and any modifications should also be a part of the software lifecycle processes and methodologies.

B.7 Custom Operating Systems.

Real-time systems may utilize a custom operating system which is designed and developed for a particular use, especially on a higher level of concern device. An executive system is essentially an operating system much like that on a personal computer that manages processes and resource allocation. They typically include a clock, an interrupt handler, a scheduler, a resource manager, a dispatcher, a configuration manager, and a fault manager. Monitoring and control systems are real time systems which are designed to a generic architecture and are used for checking sensors which provide information about the system’s environment and take actions depending on the sensors reading.

Monitoring systems take action when some exceptional sensor value is detected. Control systems continuously control hardware actuators depending on the value of associated sensors. Another type of real-time system is a data acquisition system which collect data from sensors for subsequent processing and analysis. These also typically have a generic architectural design.

B.8 Data Compression.

Many data processing devices, such as a holter monitor, involve storage of large volumes of data. To reduce the storage requirements, there is a need to reduce the redundancy in the data representation. That is, to compress the data. The compression and expansion of data can be implemented in hardware, firmware, or software. If compression and expansion is done in software, there is an increase in complexity in the software and there are many techniques that can be used to accomplish this. Data compression can be divided into two categories: irreversible and reversible. Irreversible techniques involve a reduction of the physical representation of the data, usually referred to as data compaction. All information is considered to be relevant in data compression, and the compression will be followed later by expansion which recovers the original data. It is the manufacturer's responsibility to show that the expanded data provides an accurate recount of the original data.

B.9 Embedded and Real-Time Systems.

Embedded and real-time systems include embedded software, software using a real-time operating system, programmable logic arrays (PLAs), programmable logic devices (PLDs), etc. This type of software poses unique concerns about safety and reliability because, in general, the development environment is different than the intended operational environment. Techniques and/or simulators and emulators must be employed to analyze the timing of critical events and identify non-deterministic conditions.

B.10 Human Factors and Software Design.

The focus of human factors is user interface design. Poor design induces errors and inefficiency among even the best-trained users, especially under conditions of stress, time constraints, and/or fatigue. Although labeling (e.g., user documentation) is extremely important to good performance, even well-written instructions are cumbersome to use in tandem with actual operation. Also, it's difficult to write coherent documentation which describes awkward operating procedures.

Although both hardware and software design influence the user's performance, the logical and informational characteristics provided via software are increasingly crucial. Data presented in an ambiguous, difficult-to-read, or counter-intuitive manner poses the threat of an incorrect reading, misinterpretation, and/or improper data entry. An example might be a crowded display with cryptic identifiers combined with a time lag between user response and displayed feedback. Such design characteristics overtax the user's abilities (e.g., memory, visual perception, decision-making, etc.), and resultant errors may have serious consequences.

B.10.1 Common Problems.

The logic and simplicity of control-activated operations and information access/manipulation is crucial, no matter what the program medium. Below are problem areas which lead to errors, and most are generally applicable to devices regardless of manner of control operation and information display or feedback:

- Uncertain/no feedback following input;
- missing or ambiguous prompts;
- automatic resets or defaults not initialized by the user;
- unreasonable mental calculations required;
- no query for critical input;
- complex command structure;
- unfamiliar language/coding/acronyms, mnemonics, etc;
- inconsistencies among formats for successive or co-located displays;
- conventions (e.g., color) contradictory to user stereotypes/expectations;
- ambiguous symbols or icons;
- no appropriate lock-outs or interlocks;
- illogical or cumbersome control sequences or screen call-up ("navigation");
- no status information; etc.

B.10.2 Examples.

Many user errors induced by software design are attributed to other factors due to the fact that often little, if any, physical evidence remains after the fact. Also, software-related errors can be subtle. For example, confusion from illogical data entry sequences can induce errors only indirectly related to these procedures. In any case, there are many software examples gathered from incident files, recalls, and analytic findings. Below are a few examples:

a. In at least one radiation device there have been problems due to the fact that user failure to input a dosage (time or amount?) leads to a default value. The user was not queried; nor was the default value displayed or a warning/alarm presented.

b. A neonatal monitor didn't alarm for very high heart rate. It switches to "Half-Rate" display when rate is over 240 BPM. The patient, an infant, was hypoxic and required emergency treatment.

c. CDRH discovered that a clinical batch analyzer clears all patient information fields when the operator attempts to remove any incorrect information for that patient. Also, "cleared" values are reassigned in such a way to increase the number of false negative readings over the batch.

d. There have been numerous recalls of devices in which slight deviations from prescribed operating sequences will disable the device, in some cases without any feedback to the operator.

B.10.3 Proper Analysis and Testing Pays Off.

Good human factors design involves the following; a) integrating users into the design process early; b) close coordination of software and hardware efforts; c) including user "advocates" and subject matter experts on the design team; and d) performing iterative analyses, simulations, and usability tests. Tools may involve surveys, focus tests, interviews, storyboards, documentation, etc.

The human factors engineering process can elucidate subtleties that even user-oriented designers can overlook. For example, symbols, icons, colors, abbreviations, etc. can convey a great deal of information reliably, economically, and quickly; but a priori assumptions about their meaning and clarity can be incorrect, depending upon variability among user populations, work settings, device experience, and conventions outside of the medical area. Analysis, testing and the judicious use of guidelines and standards can be incisive. In general, the human factors payoff includes fewer injuries or deaths, reduced training costs, and more marketable products. A full-length primer on human factors considerations for medical devices, titled "Do it by Design", is being prepared by FDA/CDRH Office of Health Industry Programs (OHIP). For more information about this document and its status, please contact the OHIP representatives to the CDRH Software Task Force listed on page ii.

B.11 Off-the-Shelf (OTS) Software.

Off-the-shelf (OTS) end-user software products are designed, developed, verified and validated for use in an office or industrial environment. Regularly scheduled releases of new versions of OTS software are planned which incorporate corrections and product enhancements. OTS software is not developed with the degree of rigor necessary for safety-critical applications. Hence, the responsibility for verifying and validating the use of OTS software falls to the medical device manufacturer. Verification and validation activities should evaluate the safety, reliability, and integrity of the OTS product and its intended use in a medical device, and allow for appropriate safeguards to be designed and developed for the device.

There may be instances in higher level of concern software where the use of OTS is inappropriate since the developer may not have access to appropriate documentation or source code to implement proper corrections and modifications that may be necessary, or subject the software to proper development techniques and risk management activities.

Re-engineering (or reverse engineering) is another issue with OTS software; e.g. a firm is working with only executable code and has no supporting documentation or access to source code. For higher level of concern software, finding another vendor who can support OTS software or developing a custom operating system may be a safer choice.

Use of OTS software that cannot be evaluated properly, be subjected to software lifecycle processes, or be modified if a bug or anomaly occurs, may not be appropriate to use in higher level of concern software. System level tests can be performed on OTS; however, most errors found at the system level are indicative that there are more serious problems. Errors at this level are considered symptoms, not identification of the problem.

A draft policy is being developed concerning the regulation of medical devices employing OTS software.

B.12 Open Systems and Open Systems Architecture.

Open systems may be viewed differently by many people. One view is the ability to enable dissimilar computers to exchange information and run each other's software via interfaces from independent vendors. These would be considered "open" operating systems with increased interoperability, flexibility, and portability. The idea is freeing proprietary pathways within each system. Another view is one which is used more frequently and pertains to sharing device control and communications within networks, across devices, etc. This allows for flexibility in configuring networks and systems, and may include various aspects of medical devices and hospital information systems such as intensive care units, critical care units, operating rooms, pulmonary and cardiac labs, clinical laboratories, radiology laboratories, etc. This sharing of information, control, and network time and space increases the complexity of medical devices. This may not be desirable for higher risk devices, especially since the environment will be difficult to model during verification and validation. Appropriate test suites and test cases may be difficult to analyze from a "completeness" point of view; determining when enough testing has taken place and test completion criteria has been met.

The term "open system" has typically referred to the flexibility in using several different vendor devices in a network of some kind, which would require that each vendor have appropriate knowledge of proprietary device drivers for appropriate communication. During the verification and validation process, all information needed for proper communication must be well known by all so that devices can be properly developed and tested. Because medical devices of "higher" level of concern require a more robust operating environment, open systems may not be the most appropriate approach.

B.13 Process Control Software.

Process control software is a Good Manufacturing Process (GMP), Good Laboratory Practice (GLP), or Good Clinical Practice (GCP) issue. While the same development lifecycle and risk management activities apply, the primary concern is that the software works correctly in the intended manufacturing, laboratory, or clinical process.

B.14 Redundant Displays.

Redundant displays, even if secondary, are relied upon as much as the original device monitor. A redundant display allows information to be displayed at a remote location (or different position from the parent device) and sometimes allows for limited control of the device. Manufacturers should be aware that redundant displays are considered medical devices, just as the parent device, and are reviewed as such. Therefore, software development activities for such a device should be treated with the same regard. This is especially true if the device is part of a “higher” level of concern monitoring system.

B.15 Research Shareware/Freely Distributed Software.

Research shareware is software that: 1) is developed in a university/research setting; 2) receives limited distribution in order to obtain feedback from beta testing; and 3) is developed and distributed with no intent to market. Research shareware may be distributed in the form of object code, source code, and/or source code listings. The functionality, safety features and procedures, and reliability must be validated for the intended use. Should research shareware be incorporated into a commercial product, the end manufacturer is responsible for validation, verification, and support activities. FDA/CDRH Office of Compliance, Division of Enforcement II released a legal opinion on this issue October 20, 1995, and intends to issue a notice in the Federal Register stating its position and seeking public comment.

B.16 Reuse and Libraries.

"Software reuse involves reusing existing components rather than developing them specially for an application. Systematic reuse can improve reliability, reduce management risk, and reduce development costs. Software development with reuse needs a library of reusable components that can be understood by the reuser; information on how to reuse the components should also be provided. Systematic reuse requires a properly catalogued and documented base of reusable components. Reusable software components do not simply emerge as a by-product of software development. Extra effort must be added to generalize the system components to make them reusable. Abstract data types and objects are effective encapsulators of reusable components. Development according to standards for languages, operating systems, networking and graphical user interfaces minimizes the costs of implementing a system on different types of computers." (Sommerville 1996, see Appendix E.1.)

If specifications, design documentation, test plans and procedures are written for the lowest level software components, they can be reused also. Prior to reusing software, an evaluation should be made of the appropriateness of the intended use in the new application and its affect on safety and reliability. If the development environment or lifecycle processes differ when software is reused and supporting documentation is not consistent with the new software lifecycle methodologies, some re-engineering may need to occur to import the reused software into the new environment.

B.17 Security and Privacy.

Security can be viewed in many ways. Preventing access to data or records is one view. It may also pertain to accidental or intentional data manipulation, corruption or destruction that may occur through environmental factors such as a power failure which causes data to be lost. Software should be designed, verified, and validated so that these accidental, intentional, and/or environmental data losses do not occur.

Security may also be viewed as only allowing access to records by authorized parties. For example, records that are maintained regarding anesthesia delivery and monitoring in the operating room or ICU devices that maintain patient records should not be accessible to everyone. When records are modified, an indication that the data or record was modified should appear in the record and be printed on the patient report so that it is known that someone modified the data that was recorded and retrieved by the device. Not allowing for such a modification could be a potential solution. However, with many different types of editors and data conversion programs, this is virtually impossible to assure unless proprietary encryption is used so that records would not be readable by other devices, programs, or computers. And there are some devices where data manipulation may be desirable if the user does not agree with an event marker or interpretation offered by a medical device, and may need the opportunity to override the decision on the record. It is not within the scope of this document to provide solutions to computer/software/data security issues. It is, however, in the scope of this document to raise this issue and ask that manufacturers consider this during the development of software. Some common solutions would be to provide software and data backup on a regular basis, password protection, data recovery methods, and utilize well designed and tested encryption/decryption algorithms when appropriate.

B.18 Stand-Alone Software.

A draft policy is being developed concerning the regulation of stand-alone medical software products.

B.19 Software Accessories.

Software accessories to medical devices or software that may already fall into its own regulatory classification can be placed in two categories:

a. Software that is specified/intended for use with any classified device; or

b. Software that is physically connected to a medical device for purposes of data transfer between the device and the software.

Once it is determined that software is an accessory, it does not qualify for any exemption.

B.20 User Modifiable Software.

User modifiable software includes situations where the user is able to configure a menu, the display screen, alarm and performance limits, as well as select normal values, data base information, or input their own interpretations, normal values and text. This is also a human factors issue (see B.10), but the issue of proper verification and validation during the software lifecycle becomes difficult since users can do virtually anything during use of a device. Beta site testing is important during software validation since it allows the users to use and try to intentionally break the system to ensure that proper safeguards have been incorporated. This kind of testing may be hard to duplicate off site unless users are invited to a facility during development to better facilitate the verification and validation processes. Employing appropriate requirements, device limitations, and design constraints for user modifiable software is a vital human factors and safety concern. Lifecycle processes, including testing and analysis, should account for these concerns.

Appendix C. Review Checklist and Common Requests

This appendix provides a reviewer checklist and sample questions for use in reviewing premarket medical device software submissions. These questions are frequently included in requests to manufacturers for additional information. Not all items apply to all devices; nor should these items be considered exhaustive. Appendix C is intended to be used as a memory jogger to verify the information is included in the submission. After that, the information is reviewed for content.

C.1 Review Checklist.

This checklist is intended to be used by reviewers and manufacturers to determine if the software documentation is consistent with the device.

1. Is the hazard analysis complete, and is it consistent with the device and intended use and level of concern determination?

2. Are appropriate safety requirements incorporated in the device which address the hazards identified? Have they been appropriately evaluated?

3. Are the software lifecycle processes and methodologies discussed appropriate for the safety issues and level of concern of the device? Does the submission discuss how the hazard analysis was performed?

    a. If this is a "lower" level of concern, is it adequate for addressing the hazards that were identified?

        - Are the lifecycle processes, risk control measures, and quality assurance activities reasonable for the device?

    b. If this is a "higher" level of concern, is it adequate for addressing the hazards that were identified?

        - Are the lifecycle processes, quality assurance measures, and risk management activities appropriate in addressing the safety issues for the device?

4. Are verification and validation activities performed prior to formal release?

5. Is there adequate documentation generated to assure

traceability?

6. Are configuration management and change control procedures adequate to track and control software products if corrections are necessary or modifications are made?

7. Are the software and system requirements consistent with device claims, labeling, and intended use?

8. Are potentially hazardous functions of the device appropriately monitored?

9. Are safe and unsafe operating states of the device identified and included in the warnings section of the operator's manual?

10. Are security measures consistent for the level of security required for data or device access for preventing data loss?

11. Are there redundant controls or back up mechanisms to override hazardous and non-hazardous software failures?

12. Do safety measures address component failures and environmental influences?

13. Are potentially hazardous functions of the device limited only by software? Is this appropriate?

14. Is the device architecture safe and practical for the device?

15. Is device safety assessed adequately?

16. Are the test strategies, cases, and test completion criteria sufficient to determine that the device meets its requirements, including safety?

17. Do results of tests and analyses demonstrate conformance to all requirements, including safety?

18. Are remaining software bugs non-hazardous to the patient or user? Do they impact on human factors? Are they adequately communicated to the user?

19. When data are transmitted between devices or device elements, are means provided to ensure there are no transmission errors? How is this verified?

C.2 Common Requests.

Every software product should possess safety and reliability attributes which are acquired through thorough analysis, design, implementation, testing, quality assurance, and maintenance. The software documentation generated throughout the development lifecycle should be well controlled and documented. The following should be provided in a premarket submission:

a. Please provide a composite device hazard analysis that takes into account all device hazards associated with its intended use, hardware, and software. As part of this composite analysis, please provide a software hazard analysis to demonstrate that software hazards have been considered during the software development process. Each hazard analysis should include the following:

   i. the hazardous event,
   ii. level of concern,
   iii. the method of control,
   iv. corrective measures taken, including aspects of the device design/requirements, that eliminate, reduce, or warn of a hazardous event, including a discussion of its appropriateness, and
   v. testing and evaluation demonstrating the implementation of the safety features.

   Also, please provide documentation discussing how the hazard analysis was performed, and the traceability between requirements, design, testing and risk assessment activities regarding the device hazards.

b. Please indicate if the device is of higher or lower level of concern. Include appropriate justification and analysis.

c. Please provide a discussion of your software development lifecycle processes and methodologies as it applies to the device under review. This should describe the following:

   - the lifecycle process plans and activities,
   - quality assurance plans and activities,
   - risk management plans and activities,
   - a description of the development environment,
   - design and coding standards,
   - verification and validation plans and activities,
   - the generated documentation,
   - configuration management plans and control, and
   - maintenance.

d. Please provide the software and system requirements and design. This information should include the following:

   i. hardware requirements, including microprocessors, memory devices, sensors, energy sources, safety features, device limitations, communications, etc.
   ii. programming language and program size(s),
   iii. software performance and functional requirements as follows:

      - algorithms or control characteristics for therapy, diagnosis, monitoring, alarms, analysis and interpretation (with full text references or supporting clinical data if necessary),
      - device limitations due to software,
      - internal software tests and checks,
      - error and interrupt handling,
      - fault detection, tolerance, and recovery characteristics,
      - safety requirements,
      - timing and memory requirements,
      - communication protocols,
      - identification of off-the-shelf software (if appropriate).

   iv. software and system design and architecture as follows:

      - subsystem and modularization criteria,
      - software modules, including flow and structure charts, and
      - system block diagrams.

e. Please provide the following testing and analysis information:

   i. a system level test protocol with pass/fail criteria, data, and an analysis of the results,

   ii. a verification and validation test report discussing how all phases and methods of testing and analysis (unit, module, subsystem, integration, and system) demonstrate that requirements were met. This should include all version and revision identifiers for the software and a discussion of testing results and analysis of the following (when appropriate):

      - fault, alarm, and hazard testing,
      - error, range checking, and boundary value testing,
      - timing analysis and testing,
      - special algorithms and interpretation tests and

analysis,
- path analysis and branch testing,
- stress testing,
- device options, accessories, and configurations testing,
- communications testing,
- memory utilization testing,
- qualification of off-the-shelf software (when use is appropriate),
- acceptance and beta site testing,
- regression testing, and
- test completion criteria, including test case approach and design.

iii. a fault tree analysis and failure mode effects criticality analysis of the software and explain how results were employed in the software/system requirements, design, and testing,

iv. a list of errors and bugs which remain in the device and explain how they were determined to not impact safety or effectiveness, including operator usage and human factors. These should be communicated to the user in the device labeling.

APPENDIX D. Relevant National and International Consensus Standards

The following list is a collection of current voluntary national and international consensus standards that are directly or indirectly related to medical device software safety, reliability, and lifecycle issues. The list is comprehensive but not exhaustive and is provided for consideration. It is understood that standards are continually undergoing update/reaffirmation cycles; accordingly the newest approved version should be used. The selection of a particular standard or set of standards will depend on many factors, including the design/development methodology, the type of device, the type of software, and standard corporate practices.

The standards are grouped by subject areas and may be obtained from the American National Standards Institute (ANSI) or the Institute of Electrical and Electronics Engineers, Inc. (IEEE) at the addresses below. Two special volumes should be noted: (1) ANSI publishes a complete volume of all of the ISO 9000 compendium standards; and (2) IEEE publishes a complete volume of all of their software engineering standards.

ANSI
11 West 42nd Street
New York, NY 10036
212.302.1286 (fax)
212.642.4900 (voice)

IEEE
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
908.562.1571 (fax)
908.562.3811 (voice)

D.1 General Lifecycle Activities.

1. ANSI/IEEE 1058 Standard for Software Project Management Plans, 1993.

2. ANSI/IEEE 610.12 Software Engineering Terminology, 1995.

3. ANSI/IEEE 1063 Standard for Software User Documentation, 1993.

4. ANSI/NISO Z39.67 Software Description.

5. ANSI/IEEE 1002 Standard Taxonomy for Software Engineering Standards, 1992.

6. ANSI/IEEE 1074 Standard for Developing Software Life Cycle Processes, 1991.

7. ANSI/IEEE 1016 Recommended Practice for Software Design Descriptions, 1993.

8. ANSI/IEEE 1016.1 Guide for Software Design Descriptions, 1993.

9. ANSI/IEEE 1045 Standard for Software Productivity Metrics, 1992.

10. ANSI/IEEE 830 Recommended Practice for Software Requirements Specifications, 1993.

11. ANSI/IEEE 1028 Standard for Software Reviews & Audits, 1993.

12. ANSI/IEEE 1062 Recommended Practice for Software Acquisition, 1993.

13. ANSI/IEEE 1220 Trial-use Standard for the Application and Management of the System Engineering Process, 1995.

14. ISO/IEC 12207:1995(E) Information Technology -- Software Lifecycle Processes.

D.2 Safety and Reliability.

1. Medical Electrical Equipment - Part 1: General Requirements for Safety - 4. Collateral Standard: Programmable Electrical Medical Systems, IEC (DIS) 601-1-4: 19xx.

2. ANSI/IEEE 982.1 Standard Dictionary of Measures to Produce Reliable Software, 1988.

3. ANSI/IEEE 982.2 Guide for the use of Standard Dictionary of Measures to Produce Reliable Software, 1988.

4. ANSI/IEEE 1012 Standard for Software Validation & Verification, 1992.

5. ANSI/ANS 10.4 Nuclear Computer Programs, 1987.

6. ANSI/AIAA R-013 Software Reliability, 1992.

7. ANSI/IEEE 1228 Standard for Software Safety Plans, 1993.

8. Developing safe, effective, and reliable medical software, 1991 AAMI Monograph (MDS-175). [available from AAMI, 3330 Washington Blvd., Suite 400, Arlington, VA 22201-4598, 703.276.0793 (fax), 703.525.4890 (voice)]

9. IEC 1508 Functional safety: safety related systems
   Part 1: General requirements;
   Part 2: Requirements for programmable electrical systems (PES);
   Part 3: Software requirements;
   Part 4: Definitions and abbreviations of terms;
   Part 5: Guidelines for the application of Part 1
   Part 6: Guidelines for the application of Parts 2 and 3;
   Part 7: Bibliography of techniques and measures.

10. ISO/IEC JTC1/SC7 WG9 Project 7.30 Software Integrity Levels (working draft 1.0), 1994.

11. (committee draft) IEC TC 56(secretariat)410 Dependability - Risk analysis of technological systems, 1994.

12. (committee draft) IEC SC 45A(secretariat) Control systems important to safety - 1st supplement to IEC 880, 1995.

13. IEC 812: 1985, Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA).

14. IEC 1025: 1990, Fault tree analysis (FTA).

D.3 Quality Assurance.

1. ISO 8402 Quality Management and Quality Assurance Vocabulary

2. ISO (DIS) 8402/DAM 2 Quality Management and Quality Assurance Vocabulary Amendment 2

3. ISO 9000:1987 Quality Management and Quality Assurance Standards - Guidelines for Selection and Use

4. ISO (DIS) 9000-1 Quality Management and Quality Assurance Standards - Part 1: Guidelines for Selection and Use

5. ISO (DIS) 9000-2 Quality Management and Quality Assurance Standards - Part 2: Generic Guidelines for the Application of ISO 9001, ISO 9002, and ISO 9003

6. ISO 9000-3 1991 Quality Management and Quality Assurance Standards - Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software

7. ISO 9000-4 Quality Management and Quality Assurance Standards - Part 4: Guide to Dependability Program Management

8. ISO 9001:1987 Quality Systems - Model for Quality Assurance in Design/Development, Product, Installation and Servicing

9. ISO (DIS) 9001 Quality Systems - Model for Quality Assurance in Design, Development, Production, Installation and Servicing

10. ISO 9002:1987 Quality Systems - Model for Quality Assurance in Production and Installation

11. ISO (DIS) 9002 Quality Systems - Model for Quality Assurance in Production, Installation and Servicing

12. ISO 9003:1987 Quality Systems - Model for Quality Assurance in Final Inspection and Test

13. ISO (DIS) 9003 Quality Systems - Model for Quality Assurance in Final Inspection and Test

14. ISO 9004:1987 Quality Management and Quality System Elements - Guidelines

15. ISO (DIS) 9004-1 Quality Management and Quality System Elements - Part 1: Guidelines

16. ISO 9004-2:1991 Quality Management and Quality System Elements - Part 2: Guidelines for Services

17. ISO (DIS) 9004-4 Quality Management and Quality System Elements - Part 4: Guidelines for Quality Improvement

18. ISO 10011-1:1990 Guidelines for Auditing Quality Systems - Part 1: Auditing

19. ISO 10011-2:1991 Guidelines for Auditing Quality Systems - Part 2: Qualification Criteria for Quality Systems Auditors

20. ISO 10011-3:1991 Guidelines for Auditing Quality Systems - Part 3: Management of Audit Programs

21. ISO 10012-1:1992 Quality Assurance Requirements for Measuring Equipment - Part 1: Metrological Confirmation System for Measuring Equipment

22. ISO (DIS) 10013 Guidelines for Developing Quality Manuals

23. ANSI/IEEE 730 Standard for Software Quality Assurance Plans, 1989.

24. ANSI/IEEE 1061 Standard for a Software Quality Metrics Methodology, 1992.

25. ANSI/IEEE 1298 Software Quality Management System, Part 1: Requirements, 1992.

D.4 Configuration Management.

1. ANSI/IEEE 1042 Guide to Software Configuration Management, 1993.

2. ANSI/IEEE 828 Standard for Software Configuration Plans, 1995.

3. ANSI/IEEE 1219 Standard for Software Maintenance, 1992.

D.5 Test and Evaluation.

1. ANSI/IEEE 829 Standard for Software Test Documentation, 1991.

2. ANSI/IEEE 1008 Standard for Software Unit Testing, 1993.

3. ANSI/IEEE 1044 Standard for Classification of Software Errors, Faults, and Failures, 1993.

4. ANSI/IEEE 1059 Guide for Software Verification and Validation, 1993.

D.6 Automated Tools.

1. ANSI/IEEE 990 Recommended Practice for ADA as a Program Design Language, 1992.

2 ANSI/IEEE 1175 Standard Reference Model for Computing System Tool Interconnections, 1991.

3. ANSI/IEEE 1209 Recommended Practices for the Evaluation and Selection of CASE Tools, 1992.

4. IEEE P1348 draft 6.0 (1995) Recommended Practices for the Adoption of CASE Tools.

D.7 Human Factors Engineering.

1. ANSI/AAMI HE48-1993 Human Factors Engineering, Guidelines and
Preferred Practices for the Design of Medical Devices

2. (draft) Laboratory Instruments and Data Management Systems: Design of Software User Interfaces and Software Systems Validation, Operation, and Monitoring; Proposed Guideline NCCLS GP19-P, vol. 14, no. 14, 1994. [available from NCCLS, 771 East Lancaster Avenue, Villanova, PA 19085, 610.525.2435 (voice), 610.527.8399 (fax)]

3. ANSI Z535.3-1991 Criteria for Safety Symbols.

APPENDIX E. Bibliography

This appendix cites books used in the preparation of this document and suggested additional readings on the topics covered. While this list includes many of the current publications, it is not an exhaustive list.

E.1 General Lifecycle Activities.

Blum, Bruce I. TEDIUM and the Software Process, MIT Press, Cambridge, Mass. 1990.

Blum, Bruce I. Software Engineering: A Holistic View, Oxford University Press, New York, 1992.

Budgen, David Software Design, Addison-Wesley, 1994, ISBN 0-201-54403-2.

Calvez, Jean Paul Embedded Real-Time Systems: A Specification and Design Methodology. John Wiley & Sons, 1993. ISBN 0-471-93563-8.

Edwards, Keith Real-Time Structured Methods: Systems Analysis John Wiley & Sons, 1993. ISBN-0471-93415-1.

Fairley, Richard Software Engineering Concepts, McGraw-Hill, Inc., 1985, ISBN 0-07-019902-7.

Jackson, Michael Software Requirements and Specifications, Addison-Wesley, 1995, ISBN 0-201-87712-0.

Pressman, R.S. Software Engineering: A Practitioner's Approach, McGraw-Hill, Inc., 1992.

Shumate, Ken and Keller, Marilyn. Software Specification and Design: A Disciplined Approach for Real-Time Systems. John Wiley & Sons, 1992. ISBN 0-471-53296-7.

Sommerville, Ian Software Engineering, 5th edition, Addison-Wesley, 1996, ISBN 0-201-42765-6.

van Vliet, Hans Software Engineering: Principles and Practices, John Wiley & Sons, Ltd., 1993, ISBN 0-471-93611-1.

Witt, Bernard, Baker, F. Terry, and Merritt, Everett W. Architecture and Design, Van Nostrand Reinhold, 1994, ISBN 0-442-01556-9.

E.2 Safety and Reliability.

Leveson, Nancy G. Safeware, Addison-Wesley, 1995, ISBN 0-201-11972-2.

Musa, J.D., Iannino, A. and Okumoto, K. Software Reliability Measurement, Prediction, and Application, McGraw-Hill, Inc., 1987.

Musa, J.D. Operational Profiles in Software Reliability Engineering, IEEE Software, vol. 10, no. 2, March 1993, pp. 14-32.

Neumann, Peter G. Computer-Related Risks, ACM Press/Addison Wesley, 1994. ISBN 0-201-55805-x.

Peterson, James L. Petri-Net Theory and the Modeling of Systems, Prentice-Hall, 1981.

Raheja, Dev G. Assurance Technologies, Principles and Practices, McGraw-Hill, Inc., 1991, ISBN 0-07-051212-4.

Roland, Harold E., Moriarity, Brian System Safety Engineering and Management, 2nd edition, Wiley Interscience, 1990. ISBN 0-471-61816-0.

E.3 Quality Assurance.

Cho, Chin-Kuei An Introduction to Software Quality Control John Wiley & Sons, Inc., 1980, ISBN 0-471-04704-X.

Schmauch, Charles H. ISO 9000 for Software Developers, ASQC Press 1994, ISBN 0-87389-246-1.

Schulmeyer, G. Gordon, McManus, James I. Handbook of Software Quality Assurance, 2nd edition, Van Nostrand Reinhold, 1992, ISBN 0-442-00796-5.

Schulmeyer, G. Gordon, McManus, James I. Total Quality Management for Software, Van Nostrand Reinhold, 1993, ISBN 0-442-00794-9.

E.4 Test and Evaluation.

Gilb, Tom and Graham, Dorothy Software Inspection, Addison-Wesley, 1993, ISBN 0-201-63181-4.

Habayeb, Abdul System Effectiveness, Naval Post-Graduate School.

Myers, Glenford J. The Art of Software Testing, John Wiley & Sons, Inc., 1979, ISBN 0-471-04328-1.

E.5 Human Factors Engineering.

Bias, R. and Mayhew, D. Cost Justifying Usability, Academic Press, 1994.

Brown, Martin L. Human Computer Interface Design Guidelines,Ablex Publishing Co., 1989.

Karat, C. "Cost Justifying Support on Software Development Projects", Human Factors Society Bulletin, Human Factors Society, 1992.

Norman, Donald A. The Psychology of Everyday Things, Basic Books, 1988.

E.6 FDA Publications.

These publications may be obtained from the FDA/CDRH/OHIP Division of Small Manufacturers Assistance (DSMA) at 1.800.899.0381 or 1.301.443.7491.

CBER Guideline for the Validation of Blood Establishment Computer Systems, version 1.0, October 1994.

CBER Docket No. 91N-0450, Guideline for Quality Assurance in Blood Establishments.

CDRH Blue Book Memo "Device Labeling Guidance #G91-1", dated March 8, 1991.

CDRH Deciding When to Submit a 510(k) for a Change to an Existing Device, draft #2, August 1, 1995.

CDRH FDA Policy for the Regulation of Computer Products, draft, 13 November 1989

CDRH Guidance for the Content and Review of 510(k) Notifications for Picture Archiving and Communications Systems (PACS) and Related Devices, draft August 1993.

CDRH/OHIP Do It By Design: An Introduction to Human Factors in Medical Devices, draft, February 1996.

ORA/DFI Guide to the Inspections of Software Development Activities (The Software Lifecycle), draft, November 1995.

ORA Guideline on the General Principles of Process Validation.

APPENDIX F. Glossary for Computerized System and Software Development Terminology.

The attachment to this appendix represents the current "FDA Glossary of Computerized System and Software Development Terminology" developed by the Office of Regulatory Affairs (ORA), Division of Field Investigations (DFI). This glossary may be publicly accessed and downloaded via modem:

FDA Bulletin Board 301.443.2893

parity:	no
data bits:	8
stop bits:	1
asynchronous	